Titre: Title:	Mise en oeuvre des aspects de gestion des réseaux définis par logiciels (réseaux SDN)
Auteur: Author:	Seifeddine Ben Chahed
Date:	2015
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Ben Chahed, S. (2015). Mise en oeuvre des aspects de gestion des réseaux définis par logiciels (réseaux SDN) [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1924/

UNIVERSITÉ DE MONTRÉAL

MISE EN OEUVRE DES ASPECTS DE GESTION DES RÉSEAUX DÉFINIS PAR LOGICIELS (RÉSEAUX SDN)

SEIFEDDINE BEN CHAHED
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
OCTOBRE 2015

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

MISE EN OEUVRE DES ASPECTS DE GESTION DES RÉSEAUX DÉFINIS PAR LOGICIELS (RÉSEAUX SDN)

présenté par : BEN CHAHED Seifeddine
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. CHAMBERLAND Steven, Ph. D., président
Mme BOUCHENEB Hanifa, Doctorat, membre et directrice de recherche
M. MEJRI Mohamed, Ph. D., membre

DÉDICACE

À ma maman, je t'aime.

REMERCIEMENTS

J'aimerais bien remercier ma directrice de recherche, Professeure Hanifa Boucheneb, pour ses conseils précieux, ses recommandations constructives et son aide continue pour la réalisation du présent mémoire. Je tiens à exprimer ma sincère reconnaissance à mon encadrante pour son soutien autant sur le plan humain que technique tout au long des différentes étapes de mon projet de recherche.
Je voudrais, enfin, exprimer mes remerciements aux membres de jury qui m'ont fait l'honneur de juger ce travail.

RÉSUMÉ

L'apparition des réseaux définis par logiciels a permis de faciliter la gestion des réseaux en offrant la possibilité de définir leurs politiques sous forme de programmes de contrôle. Ceci décharge les administrateurs du devoir de configurer chaque équipement à part afin d'implanter une politique donnée. Le programme de contrôle s'exécute sur un contrôleur qui se charge de le compiler pour générer les configurations nécessaires aux équipements afin de mettre en oeuvre les politiques désirées. De nos jours, les contrôleurs permettent de spécifier les aspects de gestion des réseaux sous forme de requis de haut niveau. Ce mémoire porte sur la mise en oeuvre des aspects de gestion des réseaux définis par logiciels. Nous considérons trois aspects, à savoir la composition des points d'acheminement, la garantie et la limitation de la bande passante et le placement de règles génériques. Les points d'acheminement consistent en des équipements qui permettent d'instaurer des politiques autres que le simple routage des paquets. Ces politiques peuvent inclure le contrôle d'accès ou la détection d'intrusion qui peuvent être implantés moyennant des points d'acheminement à savoir des pare-feu ou des IDS. Les applications déployées sur les réseaux peuvent exiger la garantie ou la limitation de la bande passante afin de garder des performances bien déterminées. Finalement, les règles génériques permettent de définir des politiques globales en associant des actions à des types de paquets bien déterminés. Le type d'un paquet peut être spécifié moyennant un domaine qui est défini sous forme de combinaison de conditions génériques sur les champs des entêtes de ce paquet. Étant donné que notre but est de générer les configurations nécessaires pour l'implantation des aspects de gestion spécifiés sous forme de requis, nous avons défini plusieurs méthodes qui prennent en compte les détails du réseau. La mise en oeuvre des aspects de gestion de la bande passante et de la composition des points d'acheminement a été faite moyennant un programme linéaire en nombres entiers qui prend en compte la topologie du réseau ainsi que la capacité de chaque équipement compatible SDN. D'autre part, nous avons défini trois méthodes de placement de règles génériques. Nous améliorons le temps d'exécution d'une méthode à l'autre afin de pouvoir nous adapter rapidement aux changements des politiques qui nécessitent une mise en oeuvre immédiate. Nous avons défini, en premier lieu, une méthode basée sur un programme linéaire multiobjectif en nombres entiers (MOILP). Cette méthode prend en compte les capacités des équipements compatibles SDN et la politique de routage. La deuxième méthode de placement est basée sur le calcul du flot maximum avec un cout minimum et elle permet d'améliorer le temps d'exécution avec une petite baisse des performances. Finalement, nous avons créé un algorithme glouton qui résulte d'une modification de la deuxième méthode. Cet algorithme permet de gagner

énormément en terme de temps d'exécution tout en gardant presque les mêmes performances par rapport à la deuxième méthode. Les deux dernières méthodes considèrent également les paramètres du réseau, à savoir les capacités disponibles et la politique de routage. De plus, ils permettent de placer les règles près des sources des paquets appartenant à leurs domaines afin de les traiter plus tôt. Nos méthodes de placement permettent de maximiser le nombre de règles placées dans le cas où les capacités disponibles ne permettraient pas la mise en oeuvre de toutes les règles. Les performances de nos méthodes ont été déterminées en créant un outil qui les implante. La qualité de notre outil permet également de l'intégrer facilement dans un contrôleur existant afin d'enrichir ses fonctionnalités par les aspects traités dans notre outil.

ABSTRACT

SDN is a new paradigm that reduces network management to defining control programs and deploying them on one or many entities known as controllers. Controllers communicate with the rest of SDN compatible equipments in order to install configurations that implement policies within the network. Recent advances enable the specification of networking management aspects through high level requirements regardless of the network details such as physical topology and deployed equipments. These requirements can be processed in the controllers to generate network details wise configurations that implement the requirements within the network. The purpose of this thesis is to define a set of methods that implement networking aspects within software-defined networks. We mainly consider three aspects, namely bandwidth guarantee and limitation, middleboxes composition and placement of wildcarded rules. Bandwidth management ensures performances of applications deployed within networks. Middleboxes consist in networking equipments (firewalls, IDS, load balancers...) that enable policies, going beyond simple forwarding, such as access control and intrusion detection. Wider policies may be specified by composing many middleboxes which implies traversing them in a specific order. On the other hand, network-wide policies may be described using wildcarded rules. Such rules associate actions to packets within a specific domain that can be formulated as a combination of wildcarded conditions on headers fields. The considered aspects may be used to specify a wide range of policies. We generate configurations implementing middleboxes composition and bandwidth management using an Integer Linear Program. This program considers network topology and available resources such as SDN compatible equipments capacity. Wildcarded rules placement is done in three methods. We improve execution time from one method to another as we need to fit dynamically changing policies quickly. First, we defined a multiobjective integer linear program (MOILP) that places rules while considering routing policy and available capacities within SDN compatible equipments. The second placement method is based on the computation of Minimum Cost Maximum Flow. This method allows improving significantly the execution time with a slight performances loss in comparison to the first method. Finally, we defined a greedy placement algorithm that reduces tremendously execution time and keeps nearly the same performances in comparison to the second method. Just as the MOILP based method, the two other methods consider equipments capacities and routing policy. They also tend to place rules near sources of packets within their domains. All three placement methods maximize the number of placed rules in case the available capacity is not sufficient to satisfy all the rules. We created a tool that implements all our methods and can be integrated easily

to most of the existing controllers.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

SDN	Software-Defined Networking
IP	Integer Programming
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
MOILP	Multi-Objective Integer Linear Programming
QP	Quadratic Programming
DPI	Deep Packet Inspection
QOS	Quality Of Service
API	Application Programming Interface
IDS	Intrusion Detection System
NAT	Network Address Translation
ACL	Access Control List
PBD	Pivot Bit Decomposition
RPCP	Rainbow Path Coloring Problem
SCP	Set Covering Problem
VM	Virtual Machine
CPU	Central Processing Unit
CAP	Cheapest Augmenting Path
TDD	Test Driven Development

CHAPITRE 1 INTRODUCTION

Durant ces dernières années, les réseaux informatiques classiques ont connu de grands défis qui résultent principalement des applications modernes déployées sur ces réseaux.

En effet, l'apparition du Big Data qui nécessite un traitement distribué et l'adoption de l'infonuagique par plusieurs entreprises ont induit le besoin d'avoir des réseaux dynamiques qui offrent la possibilité de s'adapter rapidement aux changements des requis.

À titre d'exemple, les usagers de l'infonuagique peuvent demander de nouvelles ressources selon leurs besoins et peuvent changer les politiques de gestion (exp : liste de contrôle d'accès, bande passante...) des infrastructures allouées. Dans d'autres mesures, les infrastructures de calcul et stockage distribués des données massives (Big Data) peuvent évoluer rapidement en adoptant de nouveaux clusters qui peuvent contenir des milliers de serveurs et de commutateurs. Ces évolutions nécessitent généralement une mise à jour des infrastructures concernées. Toutefois, la mise à jour des réseaux classiques est réalisée en configurant manuellement chaque équipement ce qui peut produire plusieurs erreurs et incohérences. De plus, ces configurations prennent beaucoup de temps ce qui ne permet pas de faire face à l'expansion rapide du monde des technologies d'information.

C'est dans ce contexte qu'est apparu le paradigme des réseaux définis par logiciels (SDN) qui permet principalement de s'adapter à la nature dynamique des applications susmentionnées.

En effet, SDN permet principalement de centraliser la logique déterminant les politiques de gestion d'un réseau dans une ou plusieurs unités appelées contrôleurs. Ces contrôleurs communiquent avec le reste des équipements du réseau via des interfaces ouvertes.

De ce fait, la définition d'une politique de gestion d'un réseau revient à écrire des programmes et les déployer dans les contrôleurs. Dans la plupart des cas, ces programmes seront compilés, en tenant compte de la topologie et des ressources disponibles, afin de générer les configurations nécessaires à chaque équipement du réseau pour mettre en oeuvre les politiques désirées.

L'architecture SDN a été adoptée par plusieurs grandes entreprises et universités à savoir, Google et Stanford. D'autre part, les fabricants des équipements pour les réseaux tels que Cisco, HP et Juniper offrent désormais des solutions SDN qui permettent de gérer les centres de données.

Notre travail permet de générer les configurations nécessaires pour mettre en oeuvre les aspects de gestion suivants :

1. La garantie et la limitation de la bande passante
2. Les politiques de compositions des points d'acheminement
3. Les politiques exprimées moyennant des règles génériques

Un réseau est principalement défini par un ensemble d'équipements qui sont connectés entre eux par des liens. Chaque lien du réseau permet de passer une quantité spécifique de donnée pendant une période bien définie. Cette quantité désigne la bande passante. Certaines applications requièrent une garantie de la bande passante pour garder de bonnes performances. Dans d'autres mesures, on aura besoin de limiter la bande passante pour certaines applications afin de ne pas saturer les liens du réseau. Le problème de la garantie et la limitation de la bande passante pour un flux de données se réduit à un problème de routage qui vise à trouver un chemin entre la source et la destination de flux qui est formé par des liens permettant la mise en oeuvre des requis sur la bande passante.

Certaines politiques de gestion d'un réseau peuvent s'exprimer sous forme de politiques de composition des points d'acheminement. Ces derniers consistent en des équipements (pare-feu, DPI...) qui mettent en oeuvre des politiques bien précises (exp : politique de contrôle d'accès pour les pare-feu).

Une politique de composition des points d'acheminement indique le fait qu'un flux de données ayant un type bien précis doit passer par une suite de points d'acheminement, dans un ordre bien défini, avant d'atteindre sa destination.

Tout comme l'aspect de la garantie et la limitation de la bande passante, la mise en oeuvre d'une politique de composition des points d'acheminement pour un flux de données se réduit à la recherche d'un routage entre la source et la destination du flux tout en passant par les points d'acheminement précisés dans la politique de composition.

Le troisième aspect de gestion permet de spécifier des politiques de gestion globales moyennant des règles génériques sans avoir besoin de connaitre les détails de réseau, à savoir la topologie et les équipements qui s'y trouvent. De ce fait, l'abstraction des détails permet de percevoir le réseau comme un grand équipement dont la gestion peut se rendre facile.

Les règles peuvent associer différentes actions aux différents types de flux de données qui circulent dans le réseau.

1.1 Objectifs de recherche

Les objectifs de la recherche sont :

1. Créer une approche qui permet de calculer le routage nécessaire pour mettre en oeuvre l'aspect de la garantie et de la limitation de la bande passante et celui de composition des points d'acheminement tout en respectant les ressources disponibles.
2. Définir une approche qui cherche un placement de règles qui permet de mettre en oeuvre une politique de gestion exprimée moyennant une liste de règles génériques. Dans le cas où les ressources disponibles ne sont pas suffisantes pour placer toutes les règles, notre approche doit maximiser le nombre de règles placées.

1.2 Plan du mémoire

Ce mémoire est constitué de six chapitres qui incluent le présent chapitre. Le deuxième chapitre fournit une présentation générale des réseaux définis par logiciel (SDN). Il présente également d'autres notions qui servent à la compréhension des approches que nous avons définies. Ces notions incluent la programmation linéaire en nombres entiers et les graphes de flots.

Le troisième chapitre passe en revue les différents travaux qui traitent la mise en oeuvre des aspects de gestion des réseaux définis par logiciel. Nous y présenterons principalement les aspects considérés par notre travail ainsi que d'autres aspects connexes.

Le quatrième chapitre porte sur les différentes approches que nous avons définies afin de mettre en oeuvre les aspects de gestion considérés. Dans une première étape, nous présenterons un programme linéaire en nombres entiers qui sert à mettre en oeuvre les politiques de composition des points d'acheminement et la gestion de la bande passante. Dans une deuxième étape, nous présenterons la mise en oeuvre de l'aspect de placement des règles à travers deux approches, à savoir un programme linéaire multiobjectif en nombres entiers et un algorithme basé sur la recherche du flot maximum avec un cout minimum.

Le cinquième chapitre a pour but de présenter l'implantation de nos approches. Nous y présenterons aussi une troisième approche rapide de placement des règles. Finalement, nous évaluerons les performances de l'outil résultant de l'implantation de nos approches.

Dans le dernier chapitre, nous conclurons le mémoire en fournissant un résumé de nos contributions et de leurs limites. Nous fournirons également quelques perspectives.

CHAPITRE 2 NOTIONS PRÉLIMINAIRES

Tout au long de ce chapitre, nous présentons les notions préliminaires nécessaires à la compréhension de la suite du présent mémoire. Nous commençons par une présentation générale du concept des réseaux définis par logiciels (SDN). Par la suite, nous présentons le modèle de programmation en nombres entiers vu qu'elle sera utilisée par la suite pour mettre en oeuvre les aspects de gestion considérés par notre travail.

Finalement, nous présentons quelques concepts de la théorie de graphe qui sont nécessaires pour comprendre les algorithmes de placement de règles. Ces concepts consistent en le couplage maximum et les graphes de flot et ils seront utilisés dans la mise en oeuvre de l'aspect de placement des règles.

2.1 Réseaux définis par logiciels (SDN)

Les infrastructures réseau courantes utilisent généralement des équipements (commutateurs, routeurs...) dont la configuration dépend des fabricants. Une fois déployées, il s'avère difficile de faire évoluer ces infrastructures en adoptant de nouveaux protocoles ou en ajoutant et supprimant des équipements. De ce fait émerge l'idée des réseaux définis par logiciels (SDN). SDN consiste en un nouveau paradigme permettant de définir le comportement des équipements du réseau moyennant un logiciel de contrôle (Shin et al., 2012). Par conséquent, SDN sépare le plan de données et le plan de contrôle. Le plan de données définit les équipements du réseau et les connexions entre eux tandis que le plan de contrôle définit le comportement de ce réseau.

La Figure 2.1 présente une vue logique de l'architecture typique d'un réseau SDN. La couche plan de données est composée principalement des équipements d'acheminement (commutateurs, routeurs...). La couche plan de contrôle est constituée principalement d'un contrôleur SDN qui permet d'héberger la logique de contrôle du réseau. Ce contrôleur met en oeuvre cette logique en accédant au plan des données à travers une interface unifiée appelée 'south-bound'. D'autre part, la couche application représente les programmes qui définissent la logique de contrôle du réseau. Ces programmes sont construits moyennant une interface de programmation appelée 'north-bound' et offerte par le contrôleur. Une application de contrôle pourrait, par exemple, définir la politique de routage ou la façon de gérer la qualité de service (QOS) dans un réseau SDN.

Comme mentionné ci-dessus, le contrôleur fournit principalement deux interfaces de program-

mation, à savoir l'interface 'south-bound' et l'interface 'north-bound'.

L'interface 'south-bound ' la plus adoptée par la communauté opérante dans SDN consiste en OpenFlow (McKeown et al., 2008). Le plan de données d'une architecture SDN, basée sur OpenFlow, est constitué principalement de commutateurs compatibles OpenFlow. De nos jours, plusieurs entreprises, telles que HP et Cisco, fabriquent ce type de commutateurs.

La spécification d'un commutateur compatible OpenFlow est standardisée par l'organisme Open Networking Foundation (ONF). Selon la spécification d'ONF (Ben et al., 2012), un tel commutateur doit contenir un ou plusieurs tableaux de flux (flow table). Chaque tableau contient plusieurs entrées (flow entries). Chaque entrée représente une règle et inclut principalement les champs suivants :

1. Domaine d'application (match fields) : ce champ définit le domaine des paquets qui sont concernés par l'entrée de tableau. Ce domaine est défini par un ensemble de conditions sur les entêtes des paquets, le port d'entrée de commutateur et d'autres métadonnées. À titre d'exemple, un domaine d'application pourrait être comme suit :

   ip_src=$cond_1$; ip_dst=$cond_2$; in_port=$cond_3$; vlan_id=$cond_4$

   Les conditions $cond_i$ sont exprimées sous forme d'une séquence de caractères binaires {0,1} et peuvent également contenir des caractères génériques (wildcards) '*'. Ce caractère peut correspondre à la fois à 0 ou 1. Chaque champ est défini sur un nombre fixe de bits. Par exemple, le champ in_port est défini sur 32 bits.
2. Priorité : Ce champ définit la priorité de la règle représentée par l'entrée du tableau.
3. Compteurs : Ces champs peuvent tenir trace de certaines données, relatives à l'entrée du tableau. Ces données peuvent inclure :
   (a) Le nombre des paquets qui ont correspondu à l'entrée qui permet de mesurer l'utilisation de cette entrée.
   (b) Le temps écoulé depuis l'ajout de l'entrée au tableau.
4. Instructions : Ces champs définissent l'ensemble d'actions à exécuter lorsqu'un paquet correspond au domaine d'application de l'entrée. Ces actions peuvent inclure :
   (a) Des actions d'acheminement à travers des ports de sorties.
   (b) Des actions de modification des champs des entêtes du paquet.
   (c) Des actions d'acheminement à travers la file mettant en oeuvre la QOS à appliquer à ce paquet.
   (d) Des actions de suppression du paquet.

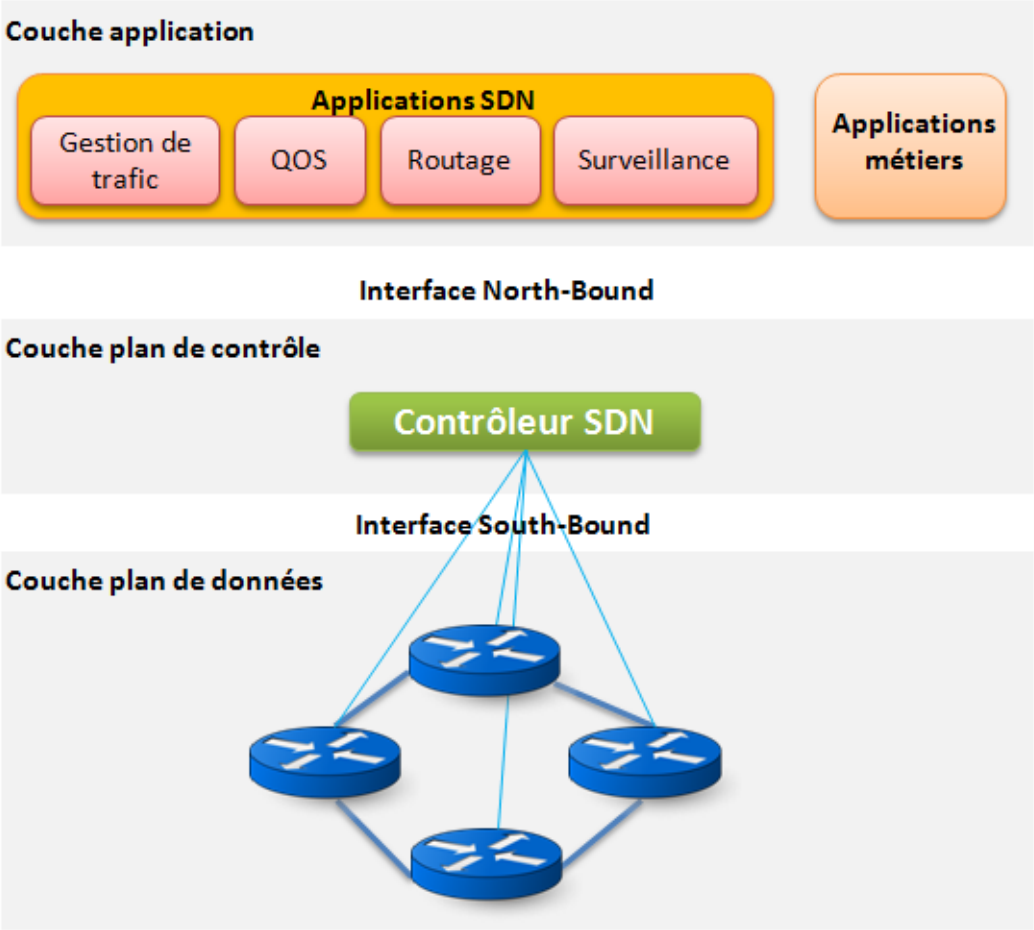


Figure 2.1 Architecture SDN

La Figure 2.2 indique le processus de traitement d'un paquet entrant dans un commutateur compatible OpenFlow et contenant un seul tableau de flux. Ce paquet sera supprimé s'il ne correspond à aucune entrée dans le tableau de flux. Dans le cas contraire, les compteurs seront mis à jour et les instructions seront exécutées.

Le contrôleur peut ajouter, modifier ou supprimer des règles dans les tableaux de flux, à travers l'API OpenFlow, d'une façon réactive (comme réponse à l'arrivée d'un paquet) ou proactive (installer les règles à l'avance dans le commutateur).

Alors qu'OpenFlow est devenu le standard de l'interface 'south-bound', l'interface 'north-bound' demeure un sujet de recherche d'actualité. Comme le montre la Figure 2.3, inspirée de (Sarwar and David, 2013), une interface 'north-bound' peut être classée selon son niveau d'abstraction et sa portée. La portée d'une interface 'north-bound' peut couvrir des environnements homogènes composés uniquement de commutateurs compatibles OpenFlow. Cette

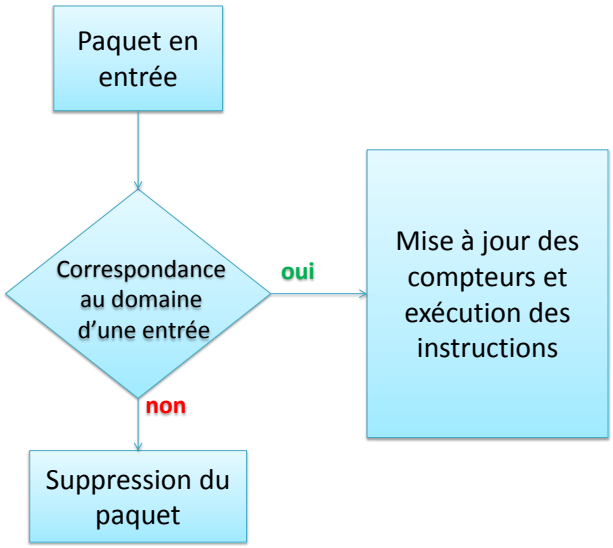


Figure 2.2 Processus de traitement d'un paquet dans un commutateur compatible OpenFlow

portée peut aussi s'étendre afin de couvrir des environnements hétérogènes contenant à la fois des commutateurs compatibles OpenFlow et des commutateurs ordinaires. Le niveau d'abstraction d'une interface 'north-bound' se mesure par sa proximité à l'API OpenFlow. En effet, une interface qui offre uniquement des opérations OpenFlow (ajout, suppression et mise à jour des règles) est considérée de bas niveau d'abstraction. Le langage NOX (Gude et al., 2008) fait partie de ces interfaces bas niveau et il offre une API OpenFlow en C++. D'autre part, Frenetic (Foster et al., 2011) consiste en un autre langage 'northbound' qui se base sur NOX pour offrir un niveau d'abstraction plus élevé permettant aux développeurs d'ignorer quelques détails lors du développement des programmes de contrôle. Frenetic offre la possibilité de construire et d'exécuter des requêtes sur l'état du réseau. À titre d'exemple, la requête suivante permet de mesurer la quantité du trafic web (port 80) qui arrive au port numéro 3 toutes les 30 secondes.

Select(sizes) *

Where(inport_fp(3) & srcport_fp(80)) *

Every(30)

Frenetic permet également la composition parallèle et séquentielle de programmes de contrôle. Cette caractéristique permet de construire la logique du contrôle du réseau en se basant sur des programmes de contrôle ayant une forte cohésion.

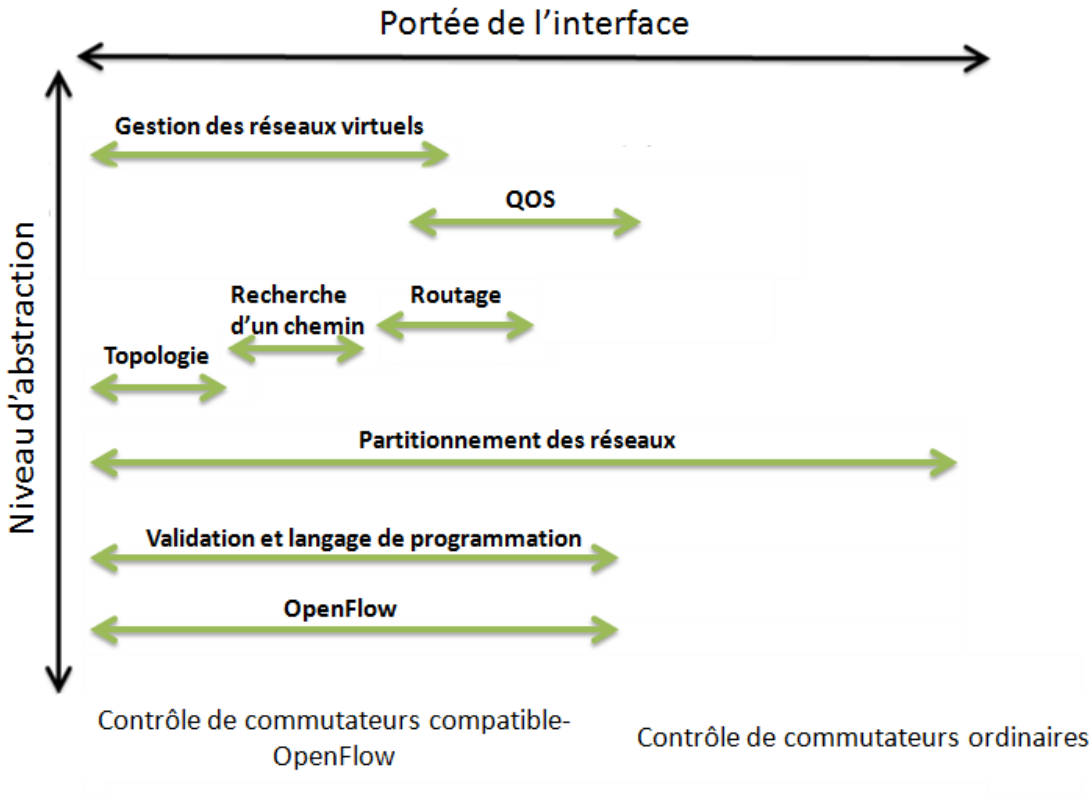


Figure 2.3 Classification des interfaces 'north-bound'

L'augmentation du niveau d'abstraction des interfaces 'north-bound' peut produire des interfaces qui permettent de gérer certains aspects de gestion des réseaux sans avoir besoin de connaître les détails d'implantation de ces aspects. Parmi ces aspects, on peut citer la qualité de service (QOS), la politique de routage, la gestion des réseaux virtuels et la gestion du trafic.

Certains de ces aspects seront étudiés au chapitre suivant qui porte sur les techniques qui mettent en oeuvre ces aspects de gestion.

2.2 Programmation en nombres entiers

Selon (Chen et al., 2010), un programme en nombres entiers (IP) consiste en un modèle d'optimisation qui se base principalement sur 3 entités.

1. Les variables : représentent les éléments qu'on désire déterminer moyennant l'optimisation. Dans le cas d'IP, ces variables sont des entiers.
2. Les contraintes : elles sont illustrées par des relations entre les variables. Ces relations

permettent de définir le domaine des variables. Ces contraintes sont généralement exprimées par des équations ou des inégalités.

3. La fonction objectif : elle consiste en une fonction qu'on veut optimiser compte tenu des contraintes.

Un programme en nombres entiers est dit linéaire si les contraintes et la fonction objectif sont linéaires. Un tel programme est dénoté par ILP. Si un ILP utilise seulement des variables binaires alors il s'agit d'un 0-1 ILP. Un programme linéaire en nombres entiers qui inclut quelques variables continues est dénoté par MILP.

Le modèle 2.1 illustre un exemple typique d'un programme linéaire en nombres entiers. Le vecteur $X = \{x_j\}_{1 \leq j \leq n}$ représente l'ensemble de variables entières. $C = \{c_j\}_{1 \leq j \leq n}$ et $A = \{a_{ij}\}_{(1 \leq i \leq m\ ;\ 1 \leq j \leq n}$ dénotent respectivement un vecteur et une matrice de coefficients. CX représente la fonction objectif à minimiser tandis que AX représente les contraintes du programme.

$$min \sum_{j=1}^{n} c_j\ x_j \tag{2.1}$$

sous les contraintes :

$$\sum_{j=1}^{n} a_{ij}\ x_j \leq\ b_i\ ; i = 1, ..., m$$

$$x_j\ \ est\ un\ entier\ ;\ j = 1, ..., n$$

Les programmes linéaires en nombres entiers permettent de modéliser beaucoup de problèmes qui débordent de la portée de la programmation linéaire classique qui utilise seulement des variables continues.

La résolution des programmes linéaires en nombres entiers est un peu différente de la résolution des programmes linéaires classiques qui peuvent utiliser des algorithmes efficaces tels que le simplexe (Dantzig et al., 1955).

De son côté, un programme ILP peut être résolu par des algorithmes qui se basent sur la résolution de la relaxation de cet ILP. La relaxation d'un programme linéaire en nombres entiers est le programme linéaire équivalent qui résulte de la transformation des variables entières en des variables continues.

Les algorithmes de résolution d'ILP les plus populaires sont :

1. 'Cutting plane' (Gomory, 1960) : Étant donné un programme ILP dénoté par P, cette méthode commence par résoudre la relaxation de ce programme. Si la solution X_s obte-

nue possède seulement des variables entières, alors elle consiste en la solution optimale. Dans le cas contraire, elle génère une contrainte 'cut' qui satisfait toutes les variables entières de X_s mais qui ne satisfait pas la totalité de cette solution, afin d'éliminer les variables continues. Ensuite, elle ajoute cette contrainte et elle répète les étapes précédentes jusqu'à l'obtention d'une solution ayant uniquement des variables entières.

2. 'Branch and bound' (Land and Doig, 1960) : Tout comme le 'cutting plane', cette méthode commence par la résolution de la relaxation du programme P. Si la solution obtenue contient des variables non entières, alors elle choisit l'une de ces variables, à savoir x_j ayant une valeur non entière λ_j. Après, elle génère les deux sous-programmes linéaires suivants :
   (a) P_1 qui est égale à P enrichi de la contrainte $x_j \leq \lambda_j$.
   (b) P_2 qui est égale à P enrichi de la contrainte $x_j \geq \lambda_j$.

Par la suite, elle explore P_1 et P_2 de la même façon que P et elle continue à explorer les sous-programmes ayant des solutions faisables et plus optimales que celles des sous-programmes, déjà explorés. La Figure 2.4 montre un exemple d'exécution de la méthode 'branch and bound'. Le programme P_1 est non faisable, alors on l'abandonne. De l'autre côté, on divise P_2 en P_{21} et P_{22}. La résolution de ces deux sous-programmes donne des solutions ayant uniquement des variables entières, cependant la solution de P_{21} est plus optimale (sa fonction objectif est inférieure à celle de P_{22}, pour un problème de minimisation).

2.3 Couplage maximum dans un graphe biparti

Selon (Fournier, 2013), un graphe biparti $G = (V,\ E)$ est un graphe dont l'ensemble de sommets V peut être divisé en deux sous-ensembles disjoints V_1 et V_2. De plus, chaque arrête $e \in E$ doit lier un sommet de V_1 à un sommet de V_2. La Figure 2.5 consiste en un exemple de graphe biparti.

Un couplage M consiste en un sous-ensemble de E tel que chaque paire d'arêtes de M ne soient pas liées au même sommet. Ce couplage M est dit maximum s'il n'existe aucun autre couplage M' tel que $|M'| > |M|$. La Figure 2.6 montre un couplage maximum qui est illustré par les arêtes colorées en vert.

2.4 Graphe de flot

Un graphe de flot est un graphe orienté $G = (V,\ E)$ avec un noeud source o et un noeud puit

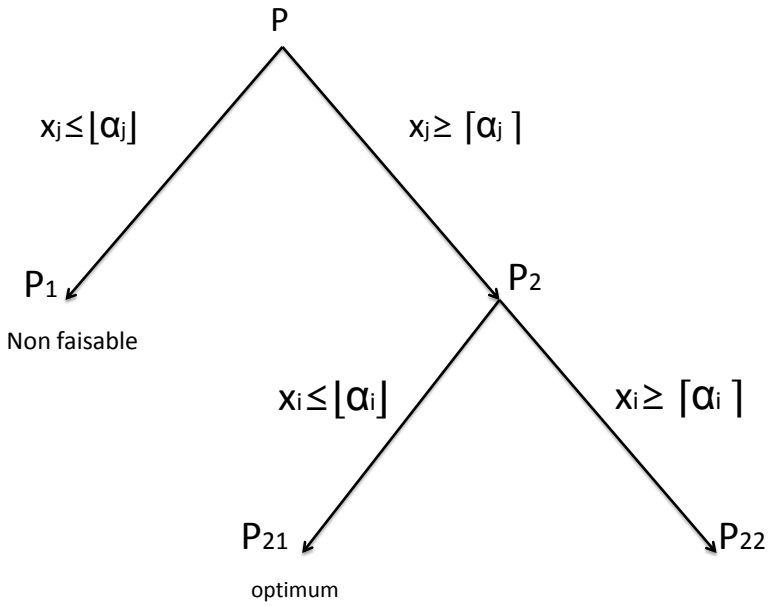


Figure 2.4 Exemple d'exécution de 'branch and bound'

d. Chaque arc e a une capacité positive dénotée par $u(e)$. La Figure 2.7 montre un graphe de flot.

Un flot peut être représenté par la fonction $f : E \rightarrow R^+$ qui obéit aux contraintes suivantes :

1. $\forall\ e\ \in\ E : f(e) \leq u(e)$
2. $\forall\ v\ \in\ V - \{o, d\} : \sum_{e \in OUT(v)}\ f(e) = \sum_{e \in IN(v)}\ f(e)$

La première contrainte indique que le flot passant par un arc e ne doit pas dépasser sa capacité $c(e)$. La deuxième contrainte indique la conservation de flot dans chaque noeud à l'exception des noeuds source et puit. La conservation de flot dans un noeud v indique que le flot rentrant est égal au flot sortant de ce noeud.

2.4.1 Problèmes basés sur le graphe de flot

Les graphes de flot sont d'une grande importance étant donné le nombre de problèmes qui peuvent être modélisés moyennant ces graphes.

Dans la suite de cette section, on présente deux problèmes importants et qui sont en l'occurrence, le problème de flot maximum et le problème de flot à coût minimum :

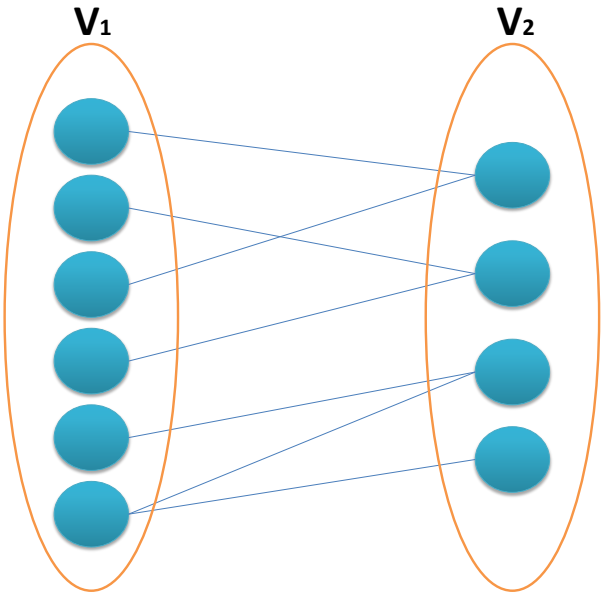


Figure 2.5 Exemple de graphe biparti

1. Le problème de flot maximum (Ford and Fulkerson, 1956) vise à maximiser la quantité de flot qui passe du noeud source o vers le noeud puit d. En faisant l'analogie entre un graphe de flot et un réseau de transportation, le problème de flot maximum est équivalent au fait de vouloir envoyer autant de véhicules d'une source vers une destination donnée, tout en sachant que les routes ne peuvent supporter qu'un nombre limité de véhicules en un temps donné. La Figure 2.8 indique le flot maximum qui passe à travers le graphe de flot de l'exemple précédent. Chaque arête est annotée par le couple $(f(e),\ u(e))$. $f(e)$ indique les unités de flot qui passent à travers e. Par exemple, le flot maximum de ce graphe est 45 et il est égal à la somme des $f(e), \forall\ e \in E$.
2. Le problème de flot à coût minimum (Klein, 1967) concerne les graphes de flot ayant plusieurs noeuds sources et plusieurs noeuds approvisionnement. Il consiste à trouver un approvisionnement, à moindre coût, d'unités de flot entre l'ensemble de noeuds sources (fournisseurs) et un ensemble de noeuds puits (demandeurs). À titre d'exemple, ce problème peut servir afin de déterminer l'approvisionnement le plus optimal d'un ensemble d'articles entre un ensemble de dépôts (fournisseurs) et un ensemble de détaillants (demandeurs). De ce fait, la définition de graphe de flot est enrichie avec la notion du coût. Chaque arc e aura, désormais, un coût par unité de flot, à savoir $c(e)$. Déterminer si un noeud $v \in V$ est demandeur ou fournisseur est fait moyennant la fonction $b(v)$. En effet, une valeur strictement positive de $b(v)$ indique que le noeud v est fournisseur

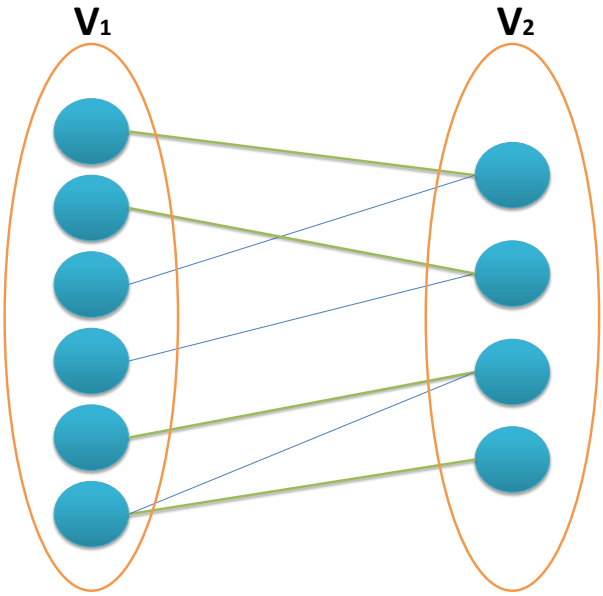


Figure 2.6 Exemple d'un couplage

tandis qu'une valeur strictement négative indique que v est demandeur.

2.4.2 Graphe résiduel

Selon (Ravindra et al., 1993), un graphe résiduel G_r est une structure qui permet de modéliser l'augmentation incrémentale du flot. À chaque étape d'augmentation, ce graphe représente la quantité du flot véhiculé et la capacité restante dans chaque arc. Soit $f(e_{ij})$ la quantité de flot qui passe à travers l'arc e_{ij}, ayant le noeud i comme source et le noeud j comme destination. Cet arc peut encore faire passer $r(e_{ij})$ unités de flot, tel que $r(e_{ij}) = u(e_{ij}) - f(e_{ij})$. De plus on peut passer $f(e_{ij})$ unités de flot sur l'arc e_{ji}, allant de j vers i. Cet arc peut servir ultérieurement pour annuler le flot $f(e_{ij})$ passant par e_{ij}. Ceci peut se faire en faisant véhiculer $f(e_{ij})$ unités de flot sur l'arc e_{ji} afin de les réorienter sur un autre arc. Dans le cas d'un graphe de flot associant des coûts $c(e)$ pour chaque arc e, l'envoi d'une unité de flot à travers un arc e_{ij} ajoutera un coût $c(e_{ij})$ aux coût total associé au flot. De ce fait, l'annulation de cette unité de flot à travers l'arc e_{ji} dois diminuer le coût $c(e_{ij})$ ajouté au coût total. Par conséquent, on associe un coût égal à $-c(e_{ij})$ à l'arc e_{ji} allant de j vers i. La Figure 2.9 présente la transformation apportée à un graphe résiduel après que $f(e_{ij})$ unités de flot soit envoyées à travers l'arc e_{ij}.

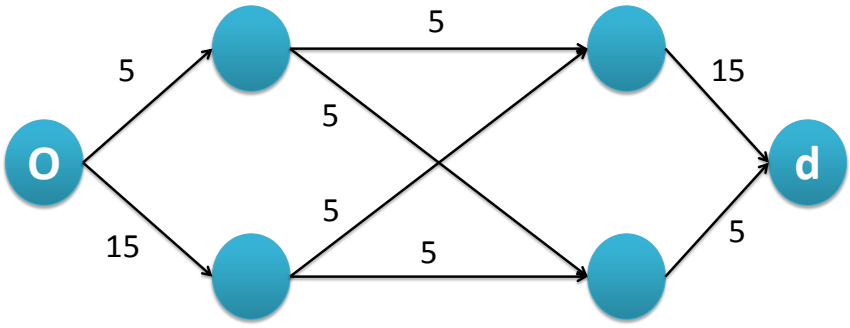


Figure 2.7 Exemple de graphe de flot

2.5 Conclusion

Ce chapitre a servi à la présentation de quelques notions qui permettent de comprendre le reste de ce mémoire. Nous y avons fourni une définition générale de SDN et la définition d'autres concepts, à savoir ILP et quelques problèmes de graphes de flot. Nous nous basons sur ces concepts afin de définir nos approches mettant en oeuvre les aspects de gestion considérés.

Dans le chapitre suivant, on va présenter et discuter les travaux connexes qui traitent les aspects de gestion des réseaux définis par logiciels.

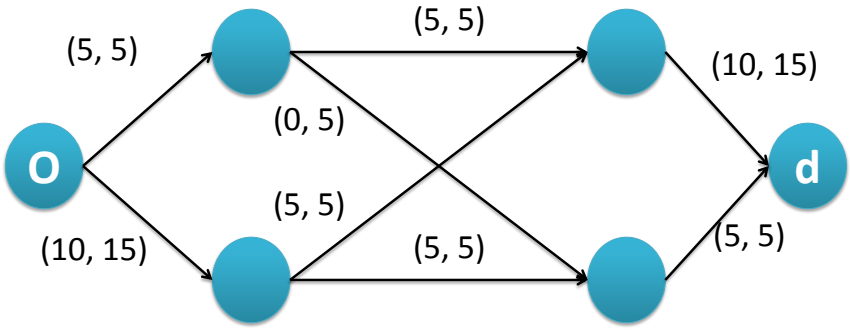


Figure 2.8 Un exemple de flot maximum

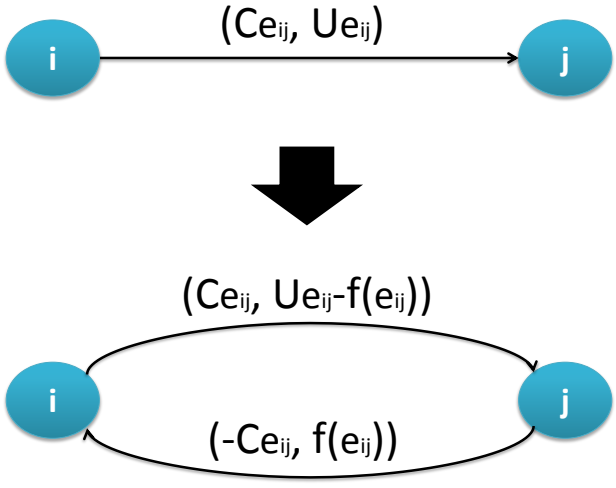


Figure 2.9 Transformation vers un arc résiduel

CHAPITRE 3 REVUE DE LITTÉRATURE

L'apparition du paradigme SDN vise principalement à faciliter la gestion des réseaux en les rendant programmables moyennant une interface de programmation (API) unifiée qui permet de surmonter les différences des interfaces des équipements déployés dans ces réseaux. Les premiers langages (Foster et al., 2011) et (Gude et al., 2008) apparus avec l'émergence de SDN offrent des API primitives dont l'utilisation ne permet pas de mettre en oeuvre facilement les aspects avancés de gestion des réseaux. Pour remédier à cette défaillance, plusieurs travaux ont introduit un niveau d'abstraction plus élevé qui permet d'exprimer les aspects de gestion des réseaux sous forme de requis ou de politiques de haut niveau. Ces derniers peuvent être exploités afin de générer automatiquement les configurations nécessaires permettant leurs mises en oeuvre dans les réseaux concernés. Tout au long de ce chapitre, nous allons principalement présenter les travaux qui portent sur la mise en oeuvre des aspects traités dans notre sujet. Les aspects de la bande passante et des politiques de composition des points d'acheminement peuvent figurer sous la même catégorie vu que leur mise en oeuvre nécessite le calcul d'un nouveau routage. Par conséquent, ce chapitre se divise en trois sections. La section 3.2 présente une revue des travaux qui mettent en oeuvre les aspects de gestion par le calcul de routage. Nous y présentons les travaux qui portent sur la mise en oeuvre des politiques de composition des points d'acheminement. Ensuite, nous y présentons les travaux qui traitent la garantie et la limitation de la bande passante. La section 3.3 fournit une revue des travaux qui mettent en oeuvre le placement des règles. Ces travaux sont classés en deux catégories, à savoir les travaux de placement qui procèdent au changement de la politique du routage et ceux qui placent les règles sans toucher à la politique du routage. Dans chacune de ces deux catégories, on traite les travaux de placement dans des réseaux homogènes et hétérogènes. Les réseaux homogènes sont composés purement de commutateurs compatibles OpenFlow tandis que les réseaux hétérogènes peuvent contenir d'autres équipements. D'autres travaux divers, touchant à d'autres aspects tels que la gestion de trafic, seront illustrés dans la section 3.4. Nous y traitons, aussi, les travaux qui portent sur la vérification des aspects de gestion dans le cadre de SDN.

3.1 Définitions

Les définitions suivantes seront valables tout au long de cette revue :

1. $N =< S,\ E >$: dénote un réseau où :

(a) S : dénote l'ensemble des équipements déployés dans le réseau N.

(b) E : dénote l'ensemble des arcs qui lient les équipements du réseau. Un arc $e \in E$ peut être appelé lien. $\forall\ u, v\ \in S$, le lien e peut être noté en tant qu'un 2-uplets (u, v). u dénote l'origine de e, tandis que v dénote sa destination.

2. $\forall\ s\ \in S$: $cap(s)$: dénote le nombre de règles qu'on peut installer dans l'équipement s.
3. $\forall\ e\ \in E$: $cap(e)$: dénote la capacité de trafic qui peut être acheminé à travers le lien e.

3.2 Aspects mises en oeuvre par le routage

3.2.1 Mise en oeuvre des politiques de composition de points d'acheminement

Les points d'acheminement (middleboxes) consistent en des équipements (ex : systèmes de détection d'intrusion, proxy...) qui permettent d'appliquer les aspects de gestion qui échappent à la portée des commutateurs compatibles SDN. La composition de ces points d'acheminement peut être utilisée pour spécifier plusieurs politiques de gestion d'un réseau donné.

Ainsi, un administrateur peut spécifier une politique de sécurité, pour un flux donné, en indiquant la séquence des points d'acheminement qui doit être traversée par ce flux avant qu'il atteigne sa destination. Par exemple, l'opérateur du réseau illustré par Figure 3.1, peut imposer à tout flux, provenant de l'extérieur de l'entreprise, de passer à travers la séquence suivante :

$$Firewall1 \rightarrow IDS1 \rightarrow Firewall2$$

La composition des points d'acheminement est rendue facile dans le cadre de SDN, étant donné la flexibilité qu'il offre. Le travail de (Qazi et al., 2013) a donné naissance à l'outil SIMPLE. Ce dernier permet de renforcer la composition des points d'acheminement pour un type de flux donné appelé classe. En prenant en compte la topologie du réseau, SIMPLE calcule la route qui permet d'acheminer le trafic de cette classe, à partir de sa source jusqu'à sa destination tout en passant, dans l'ordre, par la séquence des points d'acheminement spécifiée par sa politique de composition. De plus, SIMPLE offre la possibilité d'équilibrer la charge (load-balancing) entre deux points d'acheminement. Pour se faire, l'outil a besoin du trafic T_c associé à chaque classe c.

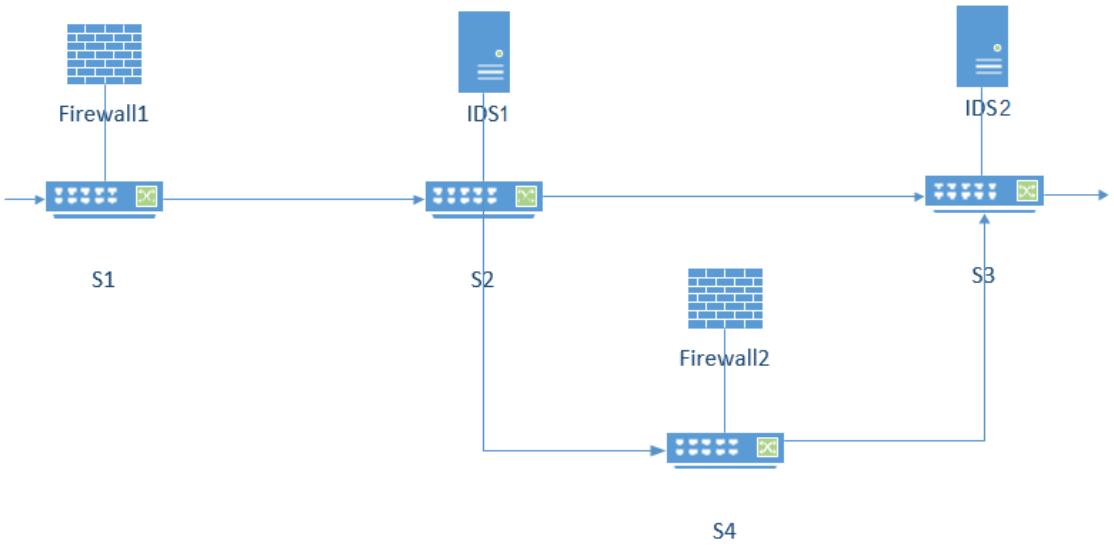


Figure 3.1 Exemple de composition

SIMPLE tient compte de la limitation des ressources utilisées dans le processus de renforcement. En effet, les auteurs ont modélisé les ressources disponibles moyennant les paramètres suivants :

1. Chaque commutateur compatible OpenFlow s_k, a une capacité $cap(s_k)$. Cette capacité exprime le nombre de règles qu'on peut installer dans s_k.
2. Chaque point d'acheminement m_j a une capacité $ProcCap_j$ qui dénote la capacité de traitement des paquets.
3. Le paramètre $Footprint_{c,j}$ dénote le coût de traitement par paquet, associé à une classe c et se trouvant dans un point d'acheminement m_j.

Comme la montre la Figure 3.2, SIMPLE est composé de trois modules :

1. "Resource Manager" : Ce module est responsable d'affecter à chaque classe c les points d'acheminement qui respectent à la fois la politique de composition et les contraintes sur les ressources. Ceci est fait en réduisant le renforcement à un problème d'optimisation qui sera décrit par la suite.
2. "Dynamics Handler" : Ce module concerne les points d'acheminement qui changent les valeurs des entêtes des paquets (ex : NAT). Cette modification de valeurs peut affecter

la sémantique de renforcement étant donné que les règles qui seront installées pour renforcer la composition sont exprimées sous forme de conditions sur les entêtes. En effet, ce module cherche la concordance entre les paquets en sortie et les paquets en entrée, moyennant un algorithme de corrélation basé sur la similarité.

3. "Rule Generator" : Ce module prend en compte les sorties des deux modules précédents afin de générer les règles qui seront installées sur les commutateurs OpenFlow déployés dans le réseau.

Avant d'entamer l'explication du problème d'optimisation utilisé par le module "Resource Manager", nous fournissons les définitions suivantes :

1. $PolicyChain_c$: dénote la séquence des points d'acheminement à suivre par les paquets de la classe c.
2. $PhysSec_c$: représente l'ensemble des séquences physiques qui peuvent être suivies par les paquets de la classe c.
3. $PhysSec_{c,q}$: dénote un élément de l'ensemble $PhysSec_c$.

À titre d'exemple : Supposons que la politique de composition d'une classe c1 est spécifiée par $PolicyChain_{c1} = \{FIREWALL - IDS\}$. Cette dernière impose que tous les paquets de la classe c1 passent, dans l'ordre, par un FIREWALL et par un IDS. L'ensemble des séquences physiques, qui permettent d'implémenter la politique de composition précédente dans le réseau de la Figure 3.1, est $PhysSec_{c1} = \{Firewall2 - IDS2;\ Firewall1 - IDS1\}$.

Le problème d'optimisation du module "Resource Manager" est décomposé en deux étapes.

1. Étape hors ligne : Étant donné une classe c et sa $PolicyChain_c$, l'objectif de cette étape est de déterminer un sous-ensemble de $PhysSec_c$ qui respecte les contraintes sur les capacités des commutateurs OF. Ceci est réalisé par le moyen d'un programme linéaire en nombres entiers (ILP).
2. Étape en ligne : Cette étape a pour but d'équilibrer la charge entre les différents $PhysSec_{c,q}$ de sous-ensemble déterminé dans l'étape précédente. Pour se faire, les auteurs ont formalisé un programme linéaire (LP) qui cherche la fraction du trafic $f_{c,q}$ qui sera alloué à chacune des $PhysSec_{c,q}$.

Outre l'outil SIMPLE, un autre article élaboré par (Soulé et al., 2014) a donné naissance à l'outil Merlin. Ce dernier renforce deux aspects qui sont en l'occurrence : les points d'acheminement et la gestion de la bande passante. Le deuxième aspect sera discuté dans la section

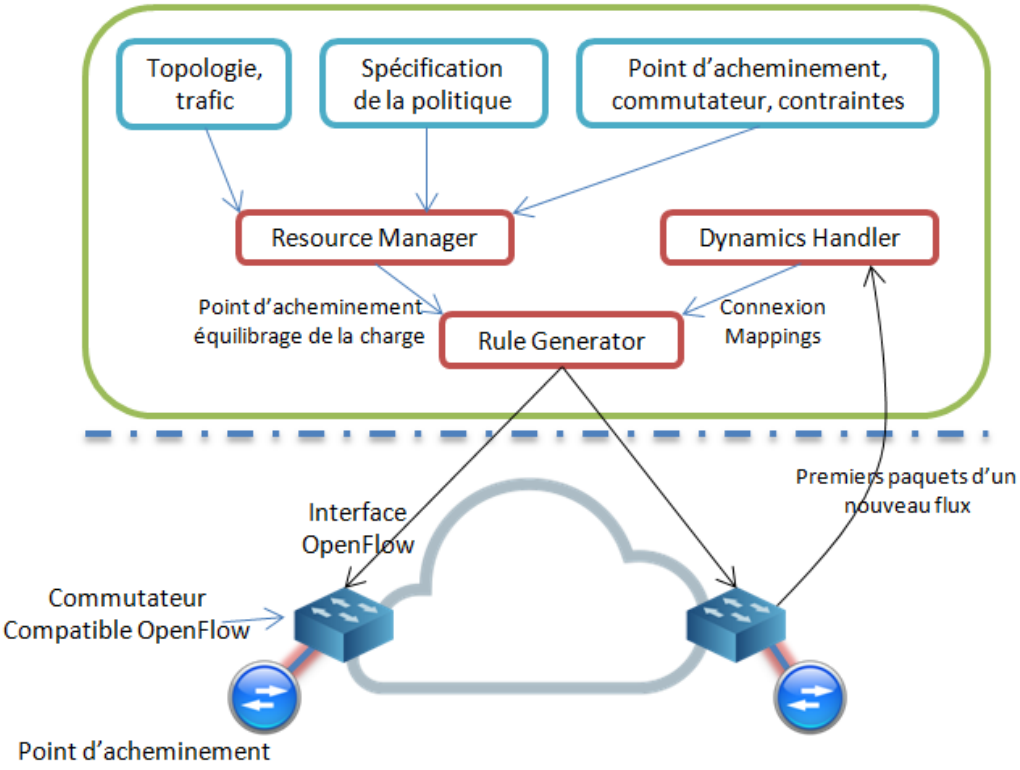


Figure 3.2 Architecture de SIMPLE

suivante. Merlin offre également un langage permettant de spécifier la politique de composition relative à un flux donné moyennant une expression régulière.

La déclaration suivante consiste en un exemple de politique de composition exprimée par le langage de Merlin. Le terme ".*" dans l'expression régulière ".* dpi *. nat .* " indique que le flux peut traverser n'importe quel nombre d'équipements avant d'atteindre le point d'acheminement spécifié par le terme qui suit.

$$[z : (ip.src = 190.165.1.1 \text{ and}$$

$$ip.dst = 190.165.1.1 \text{ and}$$

$$tcp.dst = 80) \rightarrow .* \text{ dpi } *. \text{ nat } .*]$$

De ce fait, le flux z identifié par les paramètres "ip.src, ip.dst et tcp.dst", doit traverser, dans l'ordre les points de cheminements dpi et nat avant d'atteindre sa destination finale.

Le renforcement de la politique de composition est élaboré d'une façon différente de celle de SIMPLE. En effet, Merlin construit une machine à états finis non déterministe qui accepte les instances de l'expression régulière exprimant la politique de composition d'un flux donné.

Par la suite, l'outil détermine tous les chemins physiques, qui respectent cette politique, en effectuant le produit du graphe représentant la topologie physique du réseau avec la machine à états de l'expression régulière.

La Figure 3.3 montre un exemple de construction d'un tel graphe, G_i pour l'expression régulière **"H1 .* (H1|H2|M1) .* M1 .* H2"** illustrant la politique de composition d'un flux z_i dans la topologie physique TP**.** Le produit de la topologie physique (graphe à droite) par le graphe qui représente la politique de composition (graphe au milieu), exprimée par une expression régulière, permet d'avoir tous les chemins possibles (graphe à droite) qui peuvent mettre en oeuvre la politique de composition dans la topologie. Un tel chemin est présenté en rouge dans le graphe à droite.

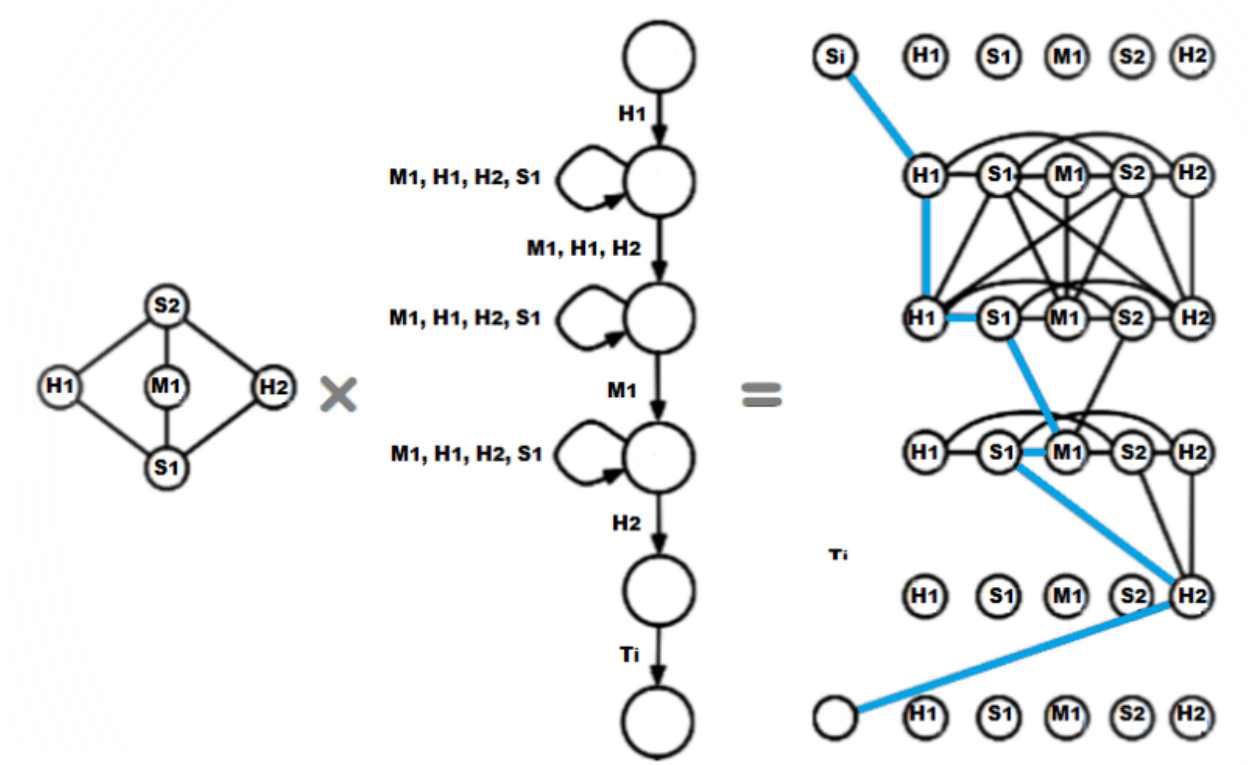


Figure 3.3 exemple de construction d'un graphe G_i

Tout en sachant que le flux z_i est caractérisé par sa source s_i et sa destination t_i. Tout chemin, reliant s_i à t_i dans le graphe G_i, renforce la politique de composition relative à z_i.

Cependant, le choix du chemin adéquat est effectué de deux façons différentes selon que le flux z_i soit restreint par le besoin de respecter des contraintes sur la bande passante ou non. En l'absence de telles contraintes, Merlin se contente d'effectuer une recherche en largeur afin

de trouver le chemin adéquat. Le cas qui traite la bande passante sera discuté dans la section suivante.

L'approche décrite précédemment renforce la politique de composition d'un seul flux. La généralisation de cette approche pour n politiques revient à construire le graphe G qui est égal à l'union des graphes $G_k(1 \leq k \leq n)$ correspondants aux différents politiques. G contient tous les chemins qui permettent de renforcer les n politiques de composition.

Tandis que SIMPLE utilise une technique de corrélation compliquée pour identifier la nature initiale des flux acheminés à travers un équipement qui modifie les entêtes des paquets. Merlin assure la rectitude de renforcement en utilisant une technique de routage basée sur les étiquettes (tag-based routing). En effet, Merlin associe une étiquette à chaque flux qui rentre dans le réseau. Les décisions d'acheminement de ce flux dans les commutateurs OF seront basées sur cette étiquette. De ce fait, les changements apportés aux entêtes des paquets constituant ce flux, n'auront aucun effet sur le routage.

Dans d'autres mesures, SIMPLE suppose que l'ensemble des chemins $PhysSec_c$, permettant de renforcer la politique de composition relative à la classe c, est déjà connu à l'avance. Cette supposition est surmontée par Merlin, qui détermine les chemins en calculant les graphes G_i. Cependant, le calcul de ces graphes pourrait entraîner une explosion combinatoire si la taille du réseau et le nombre de politiques de compositions sont importants.

Merlin a l'avantage d'offrir un langage qui permet de spécifier les politiques de compositions. Tout comme ces deux travaux, notre outil peut renforcer les politiques de compositions des points d'acheminement. Toutefois, il convient de noter que notre outil suppose qu'il existe déjà une politique de routage. De ce fait, il renforce les politiques de compositions en calculant un nouveau routage qui s'approche le plus possible du routage initial afin de minimiser le coût de reconfiguration du réseau.

3.2.2 Garantie et limitation de la bande passante

Comme indiqué ci-dessus, Merlin (Soulé et al., 2014) offre également la possibilité de renforcer des contraintes sur la bande passante. En effet, le langage fourni permet d'associer à chaque flux, la bande passante maximale ou minimale à respecter. Par exemple, la consommation en bande passante de flux z, défini dans la section précédente, peut être limitée à 25Mb/s moyennant la déclaration suivante :

max(z, 25 Mb/s)

Pour chaque flux z_i, ayant des contraintes de bande passante, Merlin cherche un chemin

allant de sa source s_i vers sa destination t_i tout en respectant ces contraintes. Ceci est réalisé moyennant un programme linéaire en nombres entiers (ILP). La formalisation de ce programme est basée sur le graphe G contenant tous les chemins conformes aux politiques de composition. L'ILP définit la variable x_e pour indiquer si le flux z_i sera acheminé ou non à travers l'arc $e \in E$ et la variable $fr_{u,v}$ afin de déterminer la fraction de $cap(u, v)$ réservée pour la garantie de la bande passante à travers le lien (u, v). $\delta^+(v)$ et $\delta^-(v)$ dénotent respectivement les arcs sortants et entrants au noeud v. r^i_{min} indique la bande passante minimum qui doit être réservée à un flux z_i. R_{max} et r_{max} indiquent respectivement la quantité et la fraction maximales réservées à la garantie de la bande passante.

Les contraintes de l'ILP sont définies comme suit :

1. $$\forall\ v \in\ G \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = \begin{cases} 1 & \text{si } v = s_i \\ -1 & \text{si } v = t_i \\ 0 & \text{sinon} \end{cases}$$

   Cette contrainte sert à trouver un chemin unique qui part de s_i pour atteindre t_i.
2. $\forall\ (u, v) \in E : fr_{u,v}\ cap(u, v) = \sum_i \sum_{e \in E_i(u,v)} r^i_{min} x_e$: Cette contrainte indique que la somme des bandes passantes minimums réservées sur un lien (u,v) est égale au produit de la fraction $fr_{u,v}$ par la capacité $cap(u, v)$.
3. $\forall\ (u, v) \in E : r_{max} \geq fr_{u,v}$: Cette contrainte assure que la fraction qui indique la capacité réservée pour la garantie de la bande passante doit être inférieur à la fraction maximale permise.
4. $\forall\ (u, v) \in E : R_{max} \geq fr_{u,v} cap(u, v)$: La bande passante réservée dans le lien (u,v) doit être inférieur à la quantité maximale permise.

Les auteurs de cet article ont défini trois fonctions objectifs pour supporter différents cas d'utilisations :

1. La première fonction objective tend à minimiser la latence en minimisant le nombre de commutateurs utilisés dans chaque chemin. Ceci est mis en oeuvre par l'expression suivante :
   $$min \sum_i \sum_{u \neq v} \sum_{e \in E_i(u,\ v)} r^i_{min} x_e$$
2. La deuxième fonction objective tend à équilibrer la charge entre les différents arcs du réseau en minimisant le r_{max} réservé dans chaque arc.

3. La troisième fonction objective sert à minimiser R_{max} afin de limiter les dégâts qui peuvent se produire dans le cas de défaillance d'un lien. Ainsi la quantité de la bande passante affectée par la défaillance sera minimum.

Tout en restant dans le même contexte, le travail de (Lee et al., 2014) permet de gérer la bande passante dans le cadre des réseaux virtuels. Un réseau virtuel est composé de plusieurs liens virtuels. L'exemple illustré par la Figure 3.4 montre l'implantation d'un réseau virtuel dans un réseau physique. Le lien virtuel (A, C) est mis en oeuvre dans le réseau physique par le chemin (A, B, C). De ce fait, cet article propose une méthode qui permet de calculer le chemin physique correspondant à chaque lien virtuel.

Le calcul de chaque chemin physique doit assurer la garantie et la limitation de la bande passante requises par le lien virtuel.

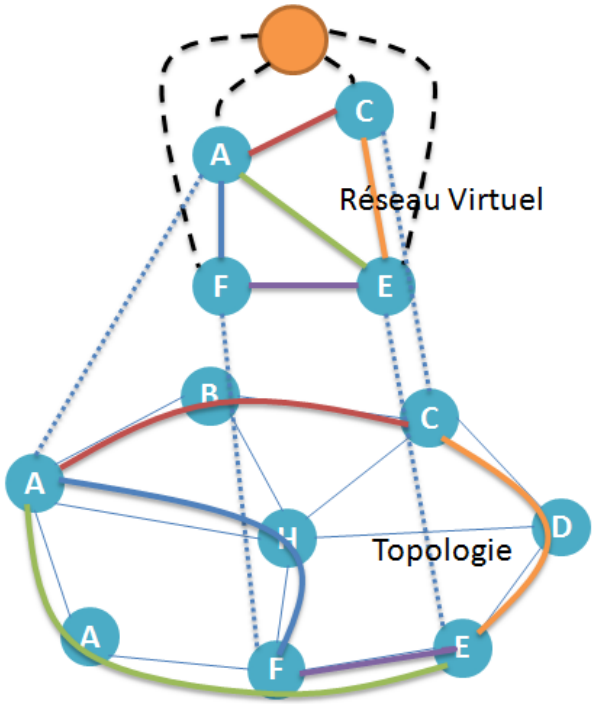


Figure 3.4 Correspondance entre un réseau virtuel et un réseau physique

Soit VL l'ensemble des liens virtuels. Pour chaque lien virtuel $vl \in VL$, on associe le couple $(u_{vl},\ g_{vl})$ qui définit les contraintes, de la bande passante, à respecter.

1. g_{vl} : indique la quantité de la bande passante qui doit être garantie pour le lien virtuel vl.

2. u_{vl} : indique la quantité de la bande passante qui ne doit pas être dépassé par le flux associé au lien virtuel vl.

On dit que le lien virtuel vl respecte les contraintes liées à sa bande passante, si la quantité de flux qui y passe dans un temps T ne dépasse pas $ba_{vl} = \frac{(u_{vl}+g_{vl})}{2}$. Ce modèle est présenté dans la Figure 3.5. On remarque que tout dépassement de la limite supérieure sera accompagné d'une suppression des paquets qui causent ce dépassement.

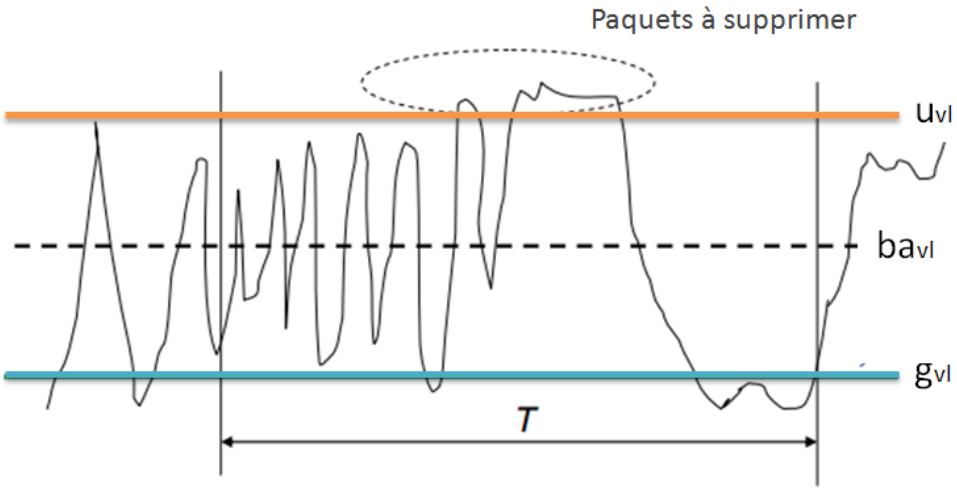


Figure 3.5 Garantie de la bande passante

Considérons les définitions suivantes :

1. D : est l'ensemble des flux à acheminer. Chaque flux correspond à un lien virtuel vl. L'acheminement de ce flux revient à trouver le chemin physique correspondant à vl. De ce fait, g_{vl}, u_{vl} et ba_{vl} se ramènent respectivement à g_d et u_d et ba_d.
2. $\forall\ d\ \in D : P_d$: désigne l'ensemble des chemins physiques qui peuvent acheminer d.
3. $\forall\ d\ \in D : \beta_d = \frac{(u_d - g_d)}{2}$.
4. $\forall\ d\ \in D : \rho_d = \frac{\beta_d}{ba_d}$.
5. $\delta_{p,\ e} = 1$ si le lien e est inclus dans le chemin physique p.
6. qd_p est une variable qui indique la quantité des demandes pour le chemin p.
7. $fr_{d,\ e}$ est une variable qui indique la quantité de la bande passante allouée pour le flux d dans le lien e.
8. α : cette variable exprime le taux maximum d'utilisation de tous les liens.

Les auteurs ont formalisé le problème de routage des liens virtuels de la façon suivante :

1. $\forall\ d \in D : \sum_{p \in P_d} qd_p = ba_d$: cette contrainte impose que tous les chemins physiques candidats doivent supporter la bande passante requise par le flux d.
2. $\forall\ d \in D,\ \forall\ e \in E : fr_{d,e} = \sum_{p \in P_d} qd_p\ \delta_{p,e}$: cette contrainte exprime la quantité qui devrait être allouée sur un lien e afin de satisfaire le flux d.
3. $\forall\ e \in E : \sum_{d \in D} (1 + \ \zeta\ \rho_d) fr_{d,\ e} \leq \alpha\ cap(e)$: cette contrainte impose le fait que la totalité de la bande passante allouée sur un lien e ne doit pas dépasser sa capacité.

Le calcul de routage a pour objectif de minimiser le taux d'utilisation de la bande passante α. Ceci permettra de subvenir aux besoins des liens virtuels en consommant le moins possible de ressources.

Les deux travaux qu'on vient de présenter opèrent presque de la même façon en essayant de renforcer les contraintes sur la bande passante moyennant un programme linéaire en nombres entiers. Néanmoins, Merlin demeure plus flexible, comme il permet de renforcer ces contraintes sur n'importe quel type de flux (ex : streaming média, transfert de fichiers...). Pour sa part, le travail de (Lee et al., 2014) se limite juste aux liens virtuels.

Notre outil offre la même flexibilité que Merlin. Cependant, tout comme le cas des points d'acheminement, notre travail considère le routage existant et essaie de minimiser la reconfiguration du réseau. En effet, notre outil renforce la bande passante et les points d'acheminement moyennant un seul programme linéaire en nombres entiers.

3.3 Méthodes de placement des règles

Les méthodes, présentées tout au long de cette section, permettent la distribution des règles à travers un réseau donné. Ces méthodes se divisent en deux grandes catégories. La première catégorie concerne les méthodes qui placent les règles sans avoir besoin de modifier la politique de routage, déjà existante. La deuxième catégorie englobe les méthodes qui modifient la politique de routage afin de placer les règles. Cette modification peut être motivée par le fait de vouloir exploiter plus de ressources pour placer les règles.

Chaque catégorie considère deux types d'environnements (réseaux) qui sont en l'occurrence, les environnements homogènes et hétérogènes. Un environnement est dit homogène s'il ne contient que des commutateurs qui peuvent supporter des règles. Toutefois, un environnement hétérogène peut contenir d'autres types d'équipements qui ne supportent pas l'installation des règles.

3.3.1 Placement sans relaxation de routage

Placement dans des environnements homogènes

L'article de (Kang et al., 2013) élaboré à l'université de Princeton, est considéré comme un travail pivot qui traite le placement des règles au sein des réseaux SDN. En effet, ce travail met en pratique un des plus grands concepts de SDN qui est en l'occurrence, l'abstraction grand commutateur (big switch abstraction). Cette abstraction considère le réseau comme un grand commutateur ayant des points de sorties (egress) et des points d'entrées (ingress). Ceci permet de spécifier des politiques globales dans le sens ou elles s'expriment en fonction des flux situés aux points d'entrées et de sorties du réseau. Un opérateur n'aura pas besoin de savoir la structure interne d'un réseau afin d'y appliquer une certaine politique.

Considérant les définitions suivantes :

1. N : un réseau ayant un ensemble de commutateurs S. Chaque commutateur $s \in S$ possède un ensemble de ports et une capacité $cap(s)$ spécifiant le nombre maximal de règles qu'il peut contenir. Le réseau possède des points d'entrées (ingress ports) et des points de sorties (egress ports)
2. pkt : un paquet dont l'entête est composé de plusieurs champs
3. loc : une location dans N qui est définie par la combinaison d'un port p_s et du commutateur s auquel il appartient.
4. EP : politique globale
5. $Route$: politique de routage définie par la fonction $Route(loc_1,\ loc_2,\ pkt)$ qui est égale à la suite des commutateurs que traverse pkt, dans l'ordre, pour aller de loc_1 vers loc_2.

Une politique globale EP s'exprime sous forme d'un ensemble de règles classées par ordre de priorité et utilisant le caractère générique (*) (wildcarded rules). La liste de Figure 3.6 constitue un exemple de règles spécifiant une politique de contrôle d'accès.

Cet article propose un algorithme qui renforce la politique EP au sein de réseau N tout en considérant le routage $Route$. Renforcer EP revient à trouver une distribution de ses règles sur l'ensemble des commutateurs S tout en respectant leurs capacités. Pour ce faire, les auteurs ont décomposé le problème selon les chemins P_i définis par la politique de routage $Route$. En effet, ils considèrent que chaque chemin P_i est associé à un espace de flux D_i qui définit l'ensemble des paquets (pkt) qui peuvent être acheminés à travers ce chemin.

R1 : (src_ip = 0*, dst_ip = 00 : Accepter)
R2 : (src_ip = 01, dst_ip = 1* : Accepter)
R3 : (src_ip = *, dst_ip = 11 : Supprimer)
R4 : (src_ip = 11, dst_ip = * : Accepter)
R5 : (src_ip = 10, dst_ip = 0* : Accepter)
R6 : (src_ip = *, dst_ip = * : Supprimer)

Figure 3.6 Exemple de règles

La décomposition du problème consiste donc à considérer chaque chemin P_i afin d'y distribuer le sous-ensemble de EP qui concerne l'espace des flux D_i. Ce sous-ensemble, désigné par EP_i, est appelé la projection de EP sur D_i.

La distribution des règles de EP_i sur le chemin P_i pourrait se faire moyennant l'algorithme de la Figure 3.7 :

Néanmoins, les chemins peuvent avoir des commutateurs en commun comme le montre l'exemple suivant :

1. $P_1 = s_1,\ s_2,\ s_3,\ s_5$
2. $P_2 = s_4,\ s_3,\ s_6$

Afin de trouver une répartition équitable de la capacité, disponible sur chaque commutateur, entre les différents EP_i, les auteurs ont introduit une étape préliminaire qui sert à diviser cette capacité. Par conséquent, la répartition de chaque sous-politique EP_i sur le chemin correspondant P_i va devoir exploiter, seulement, la capacité réservée à cette sous-politique.

L'allocation des capacités est réalisée moyennant un programme linéaire. Étant donné la

```
1 placement(EP_i , P_i ){
2       pour chaque s_k de P_i {
3               EP_ij = extraire_portion(EP_i , cap(s_k));
4               pour chaque règle r_i,j,l de EP_ij {
5                       placer r_i,j,l dans s_k ;
6               }
7               EP_i = EP_i – EP_i,j ;
7       }
8 }
```

Figure 3.7 Algorithme de placement de EP_i dans les commutateurs du chemin P_i

variable $por_{i,j}$ qui indique la portion de $cap(s_j)$ alloué à EP_i, le programme linéaire essaie de trouver ces portions tout en respectant les deux contraintes suivantes :

1. la somme des portions allouées dans un commutateur s_i ne doit pas dépasser sa capacité totale
2. la somme des portions allouées à une sous-politique EP_i doit être supérieure ou égale à la capacité totale requise par cette sous-politique.

La capacité totale requise par EP_i, dénotée par η_i est estimée de la façon suivante :

$$\eta_i = (nombre\ des\ règles\ contenues\ dans\ EP_i) * (longueur\ de\ P_i)$$

Les auteurs essaient de placer les règles de chaque EP_i le plus proche possibles des débuts des chemins P_i. Supposons que EP_i contient une règle qui impose la suppression d'un flux z_i traversant P_i. Placer cette règle au début de P_i, va permettre de diminuer le nombre de commutateurs à traverser avant que z_i soit supprimé. Cette préférence de placement aux débuts des P_i est formalisée dans l'objectif du programme linéaire précédant.

Compte tenu de la division des capacités par le programme linéaire, la 3e ligne de l'algorithme de placement est transformée comme suit :

3 $EP_{i,j}$= extraire_portion(EP_i, $por_{i,k}$ $cap(s_k)$) ;

Cet article considère que les sous-politiques EP_i sont représentées moyennant un espace des entêtes. À titre d'exemple, l'espace représentant la liste des règles de la Figure 3.6 est illustré dans la Figure 3.8. Pour faciliter la compréhension, on traite le cas des espaces bidimensionnels.

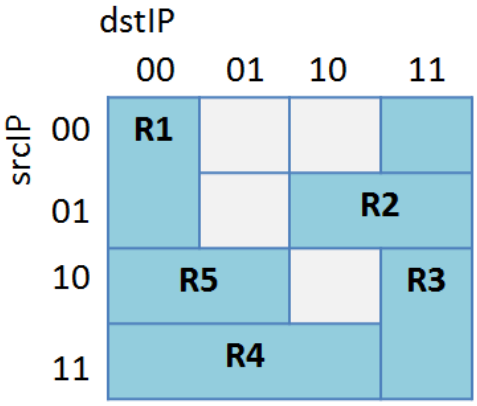


Figure 3.8 Exemple d'un espace des entêtes

De ce fait, la fonction “extraire_portion(sous-politique, taille_de_la_portion)” est définie de la façon suivante : étant donné l'espace représentant EP_i, l'algorithme essaie de couvrir une partie de cet espace moyennant un rectangle q comme la montre Figure 3.9.

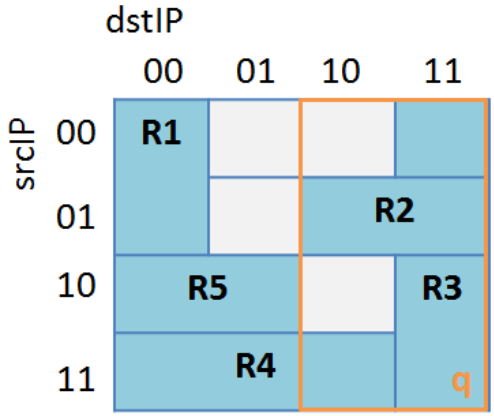


Figure 3.9 Exemple de couverture dans l'un espace des entêtes

Le nombre de règles couvertes par le rectangle q ne doit pas excéder le paramètre taille_de _la_portion. Outre la contrainte de taille, le choix de q doit minimiser le chevauchement des règles. Une règle est dite chevauchée si elle est coupée par le rectangle q. La règle R4 de la Figure 3.9 constitue un exemple de règle chevauchée. Étant donné que le contenu de chaque rectangle sera placé dans un seul commutateur, la division d'une règle en deux nécessite deux unités de capacité pour la placer (une unité dans chaque commutateur).

Pour minimiser la division des règles, les auteurs ont défini la fonction suivante qui dénote l'utilité d'un rectangle q donné :

$$utilité\ (q)\ = (nombre\ de\ \ règles\ internes\ - 1)\ \div\ (nombre\ de\ règles\ qui\ chevauchent)$$

De ce fait, l'algorithme procède à une recherche qui commence par un rectangle qui couvre tout l'espace, afin de trouver le rectangle ayant la valeur la plus élevée de la fonction utilité.

Tout en restant dans le même contexte, le travail de (Zhang et al., 2014), faisant l'objet d'une collaboration entre l'université de Princeton, Google et NEC Labs, a proposé une méthode de placement de règles qui se base sur un programme linéaire en nombres entiers (ILP). Cet article se concentre sur le placement des règles de contrôle d'accès (ACL) tout en respectant les contraintes de capacité des différents commutateurs déployés dans le réseau.

Considérant les définitions suivantes :

1. L : dénote l'ensemble des points de sortie et d'entrée du réseau N. l_i est un élément de

cet ensemble.

2. LP_i : dénote l'ensemble des chemins qui partent du point l_i. Chaque chemin P_j de LP_i est représenté par une suite de commutateurs s_k.
3. S_i : dénote l'ensemble des commutateurs utilisé dans tous les chemins de LP_i.
4. Q_i : dénote la politique associée à un point d'entrée du réseau. Cette politique est exprimée moyennant une liste de règles $r_{i,\ j}$.

Comme le montre la Figure 3.10, le processus de placement commence par des étapes préliminaires qui servent à la suppression des redondances et la fusion des règles. La fusion des règles permet d'écrire plusieurs règles, appartenant à plusieurs politiques, sous forme d'une seule règle afin de réduire l'espace consommé. Deux règles, appartenant à deux politiques différentes, peuvent être fusionnées si elles correspondent aux mêmes flux et si elles ont les mêmes actions.

Le programme linéaire en nombres entiers est défini comme suit. La variable $v_{i,\ j,\ k}$ est une variable binaire qui indique si la règle $r_{i,\ j}$ est placée dans le commutateur s_k.

Les auteurs ont défini trois contraintes :

1. La première contrainte assure la cohérence des règles en se basant sur leurs priorités. Étant donné qu'on traite uniquement des règles de type ACL, les actions spécifiées par ces règles ne doivent pas se contrarier. Par exemple, une règle déployée ne doit pas permettre la suppression d'un paquet qui a été accepté par une autre règle plus prioritaire. Pour se faire, la contrainte spécifie que si une règle $r_{i,\ w}$, ayant une action 'SUPPRIMER', est placée dans un commutateur s_k alors toute règle $r_{i,\ u}$ ayant une action 'ACCEPTER' et une priorité plus haute que celle de $r_{i,\ w}$, doit être placé dans le même commutateur.
2. La deuxième contrainte indique que le placement de chaque règle, ayant une action 'SUPPRIMER', est obligatoire.
3. La troisième contrainte indique que le nombre de règles installées sur un commutateur s_k ne doit pas dépasser sa capacité $cap(s_k)$.

Les auteurs ont donné la possibilité de choisir entre deux fonctions objectif selon le besoin. La première fonction vise à minimiser le nombre de placements effectués afin de renforcer toute la politique. La deuxième fonction essaie de placer les règles le plus proche possible de leur point d'entrée afin de minimiser le trafic circulant sur le réseau.

Un commutateur peut contenir des règles provenant de plusieurs politiques Q_i. Ceci pose un problème quand à la règle à appliquer si le flux en entrée correspond à la fois à plusieurs règles appartenant à des politiques différentes. Afin de distinguer la politique Q_i à appliquer pour un flux donné, les auteurs utilisent une technique d'étiquetage qui sert à différencier ces politiques au sein de chaque commutateur. En effet, chaque flux, qui rentre dans le réseau, sera étiqueté pour indiquer son point d'entrée. L'étiquette spécifiant une politique Q_i sera ajoutée, comme condition de correspondance, aux règles de cette politique. Le mécanisme d'étiquetage prend en compte les règles, résultant de la fusion de plusieurs règles, en leur associant des étiquettes différentes.

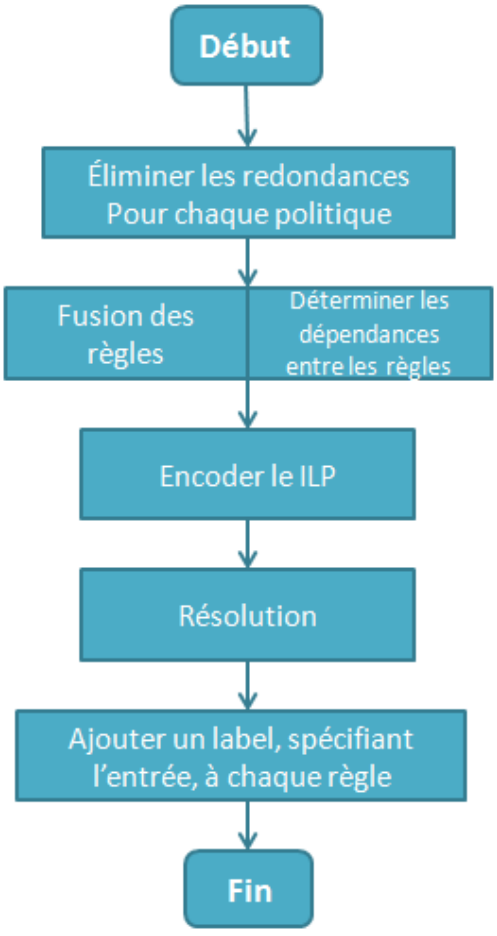


Figure 3.10 Processus de placement des règles ACL

En résumé, les deux articles présentés ci-dessus permettent de placer des règles dans des réseaux homogènes. Pour ce faire, l'article de (Kang et al., 2013) utilise un algorithme glouton tandis que l'article de (Zhang et al., 2014) modélise le problème de placement sous forme

d'un ILP.

Le premier article est plus générique vu qu'il néglige la sémantique des règles tandis que le deuxième se concentre uniquement sur les règles de contrôle d'accès. Il faut noter que les deux travaux permettent de respecter les contraintes sur les capacités des commutateurs. Cependant, aucun d'entre eux ne considère le cas où la totalité de l'espace disponible dans les commutateurs ne parvient pas à satisfaire toutes les règles. Les deux travaux échouent si les ressources, disponibles dans les commutateurs, ne sont pas suffisantes.

Notre outil, par contre, maximise le nombre de règles placées lorsqu'il n'y a pas assez d'espace dans les commutateurs. Par conséquent, notre outil réussit toujours à placer des règles.

Placement dans des environnements hétérogènes

L'outil Palette illustré par l'article de (Kanizo et al., 2013) peut s'intégrer sous cette catégorie. Étant donné une politique illustrée par un large tableau de règles, Palette renforce cette politique en décomposant ce tableau en de petits tableaux qui seront répartis sur l'ensemble des commutateurs d'une façon qui respecte leurs capacités.

Les auteurs ont proposé plusieurs méthodes de décomposition parmi lesquelles on peut citer la décomposition selon le bit pivot (Pivot Bit Decomposition - PBD). Considérons le Tableau 3.1, illustrant une politique initiale. Chaque ligne φ_i représente le domaine d'application, exprimé sur 7 bits dans le cas présent, d'une règle r_i. Par exemple, le domaine d'application φ_1 de la règle r_1 est '*010*00'.

Tableau 3.1 Exemple de politique de départ

	0	1	2	3	4	5	6
φ_1	*	0	1	0	*	0	0
φ_2	0	*	1	*	*	*	0
φ_3	*	1	*	*	1	0	1
φ_4	1	1	1	*	1	*	*
φ_5	1	1	*	0	*	*	*
φ_6	1	0	0	1	0	1	*
φ_7	*	*	*	*	*	*	*

La méthode PBD décompose ce tableau d'une façon itérative. À chaque itération, on choisit une colonne, désignant un bit pivot, du tableau. Cette colonne est utilisée afin de décomposer le tableau en deux sous-tableaux. Le premier sous-tableau contient les règles dont le bit pivot est égal à 0, tandis que le deuxième contient les règles dont le bit pivot est égal à 1. Les règles

ayant un bit pivot spécifié par le caractère générique '*' seront dupliquées dans les deux sous-tableaux. En effet, le caractère générique sera replacé par 0 dans le premier sous-tableau et par 1 dans le deuxième. Les Tableaux 3.2 et 3.3 montrent une décomposition, selon le bit pivot numéro 1, du Tableau 3.1.

Tableau 3.2 Résultats de décomposition contenant les règles ayant la valeur 0 affectée au champ 1

	0	1	2	3	4	5	6
φ_1	*	0	1	0	*	0	0
φ_2	0	0	1	*	*	*	0
φ_6	1	0	0	1	0	1	*
φ_7	*	0	*	*	*	*	*

Tableau 3.3 Résultats de décomposition contenant les règles ayant la valeur 1 affectée au champ 1

	0	1	2	3	4	5	6
φ_2	0	1	1	*	*	*	0
φ_3	*	1	*	*	1	0	1
φ_4	1	1	1	*	1	*	*
φ_5	1	1	*	0	*	*	*
φ_7	*	1	*	*	*	*	*

Le choix de bit pivot doit réduire la taille des sous-tableaux résultants par rapport à la taille du tableau faisant l'objet de la décomposition. Le PBD s'arrête lorsqu'il n'y aura plus de réduction possible ou lorsqu'on atteint un nombre de sous-tableaux spécifié à l'avance.

Étant donné les sous-tableaux résultant de la décomposition, les auteurs proposent un algorithme de distribution qui vise à répartir ces sous-tableaux sur les commutateurs de réseau. Pour ce faire, l'algorithme de distribution est transformé en un problème de coloration de chemin arc-en-ciel (Rainbow Path Coloring Problem - RPCP) en supposant que chaque sous-tableau correspond à une couleur unique et que le nombre de sous-tableaux (nombre de couleur) est égal à c.

P représente l'ensemble des chemins définis par la politique de routage de réseau. Chaque chemin P_i est représenté par une séquence des sommets $\{s_k\}$.

Le but de RPCP est de colorier chaque sommet par une couleur parmi les c couleurs, tout en considérant que les sommets de chaque chemin P_i doivent contenir les c couleurs. Autrement

dit, chaque commutateur doit contenir au maximum un sous-tableau et chaque chemin P_i doit contenir tous les sous-tableaux. La résolution de ce problème est faite moyennant un algorithme glouton.

L'algorithme de répartition est étendu au cas d'un réseau hétérogène. Ceci implique qu'on peut avoir différents types d'équipements qui peuvent ou non contenir des règles. Pour se faire, les auteurs ont modifié l'algorithme pour permettre à chaque sommet s_i de supporter d_i couleurs. d_i peut être égale à 0 dans le cas des équipements qui ne peuvent pas contenir des règles. Il peut également être supérieur à 1 si l'on suppose que l'équipement peut contenir plusieurs sous-tableaux.

Le travail de (Wang et al., 2014) traite le contexte où on ne peut placer les règles que dans certains équipements, appelés points d'application de décision (policy enforcement points). De ce fait, les auteurs de cet article formalisent le problème de placement des règles de la façon suivante : en partant d'une politique de sécurité SP, illustrée par des règles de contrôle d'accès, et d'un ensemble de points d'application $AP \subseteq N$, le problème de placement se réduit à la recherche du plus petit sous-ensemble de AP qui permet de renforcer la politique de sécurité en effectuant des inspections sur les chemins de données.

Les règles de contrôle d'accès sont modélisées moyennant l'espace des entêtes (header space). Les auteurs étendent ce concept pour définir la notion de l'espace de la politique (Policy Space) qui est l'espace contenant toutes les règles qui représentent la politique de sécurité. Les définitions suivantes illustrent les éléments du problème moyennant le concept de 'Header Space' : S_r : l'espace d'une règle modélisant une règle $r \in SP$. S_p : l'espace couvert par la politique de sécurité modélisant la politique SP. S_f : l'espace couvert par un point d'application de décision modélisant l'ensemble de règles qui peuvent être placées dans ce point .

Le renforcement de la politique de sécurité se ramène à un problème de couverture par ensembles (set covering problem-SCP). Étant donné un ensemble d'éléments U, appelé l'univers, et un ensemble V contenant n ensembles d'éléments $EV_i \subseteq U$ tel que $EV_1 \cup ... \cup EV_n = U$, le SCP consiste à chercher le plus petit sous-ensemble $C \subseteq V$ dont l'union de ces éléments EV_k est égale à l'univers. On dit que C couvre U. À titre d'exemple, supposons que $U = \{4, 5, 6, 8, 9\}$ et $V = \{\{4, 5\}, \{5, 6, 8\}, \{4, 9\}\}$, le plus petit sous-ensemble de V couvrant U est $C = \{\{5, 6, 8\}, \{4, 9\}\}$. Dans le cadre de cet article, l'univers U est représenté par l'espace S_p tandis que l'ensemble V est représenté par l'ensemble des espaces S_f. Par conséquent, renforcer la politique de la sécurité revient à chercher le plus petit sous-ensemble de S_f qui couvre S_p.

```
00  Enforce_Policy(nodes, sf_list, sr_list):
01      sf_rem ← sp_rem ← empty
02      for sf in sf_list:
03          sf_rem ← sf_rem.union(sf)
04      for sr in sr_list:
05          sp_rem ← sp_rem.union(sr)
06      sp_rem ← sp_rem.intersect(sf_rem)
07      while sp_rem:
08          n ← Select_Node(nodes, sf_list, sr_list, sf_rem, sp_rem)
09          n_sf ← sf_list[n]
10          n_sf_rem ← n_sf.intersect(sf_rem)
11          n_bypass ← n_sf.subtract(sf_rem)
12          n_enforce ← empty
13          for sr in sr_list:
14              if sr.is_intersected(n_sf_rem):
15                  n_enforce.append(sr)
16          sf_rem ← sf_rem.subtract(n_sf_rem)
17          sp_rem ← sp_rem.subtract(n_sf_rem)
18          nodes.remove(n)
```

Figure 3.11 Algorithme de renforcement de la sécurité

L'algorithme de renforcement représenté dans la Figure 3.11 opère d'une façon gloutonne. En effet, il choisit à chaque étape (ligne 8) le point d'application, dont l'espace associé S_f couvre la majeure partie des règles contenues dans l'espace de la politique qui reste à couvrir (Sp_rem). Ensuite, il place les règles correspondantes dans le point d'application courant (ligne 13). Ceci est répété jusqu'à ce que tout l'espace de la politique soit couvert ou, autrement dit, jusqu'à ce que toutes les règles formant cet espace soient placées.

Une re-exécution de l'algorithme sera nécessaire quand on veut ajouter, supprimer ou mettre à jour de nouvelles règles.

Le travail de (Moshref et al., 2013) a entraîné l'apparition de l'outil vCRIB. Cet outil permet de gérer les règles dans le cadre de l'infonuagique (cloud computing). En effet, une architecture infonuagique peut contenir plusieurs types d'équipements ayant des ressources limitées. Par exemple, la Figure 3.12 montre une architecture typique qui contient des hyperviseurs $\{HS_i\}_{1=i=6}$ et des commutateurs (ToRi ; Aggj). Chaque hyperviseur est contraint par une puissance de calcul (CPU) tandis que chaque commutateur est contraint par une capacité (memory).

vCRIB permet de distribuer un ensemble de règles, spécifiant une politique quelconque, sur l'ensemble des équipements (hyperviseurs et commutateurs) contenus dans le réseau tout en

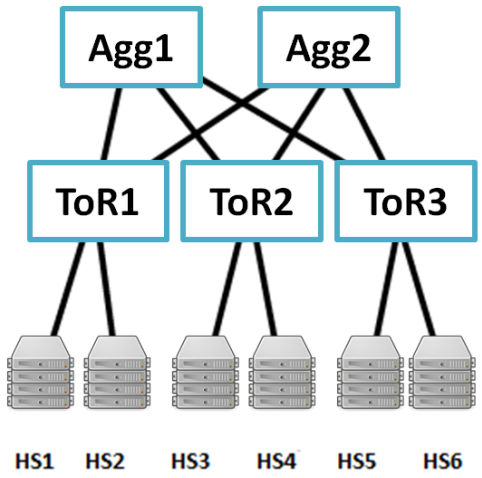
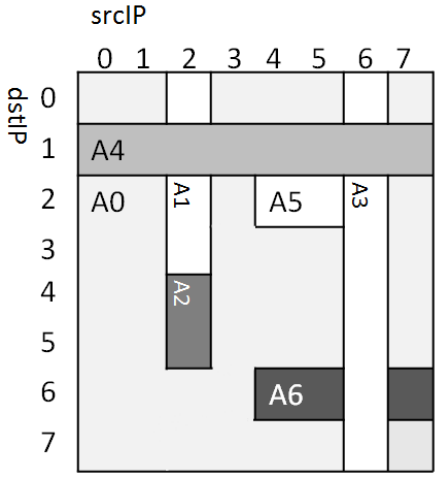
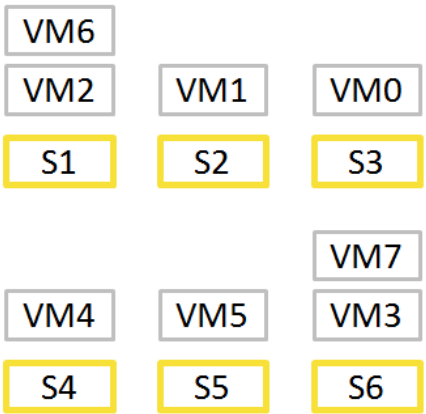


Figure 3.12 Exemple d'architecture infonuagique

respectant les ressources disponibles. La Figure 3.13a montre un ensemble de règles ACL représentées moyennant un espace de flux. Cet espace est défini sur deux axes : SrcIP et DstIP, représentant les adresses des machines virtuelles (VM) déployées dans les hyperviseurs. La règle A2, par exemple, indique que chaque flux provenant de VM2 ne doit pas être acheminé vers VM4 et VM5. La Figure 3.13b montre la distribution des VMi sur l'ensemble des hyperviseurs.



(a) Distribution des règles



(b) Distribution des VMi

Figure 3.13 Exemple de distribution des règles et des VMi

Le processus de placement des règles commence par la division de l'espace des flux en des partitions. Chaque partition doit contenir un ensemble de règles qui peuvent être placées dans le même équipement. Comme le montre la Figure 3.14, les auteurs ont fait le choix de

diviser l'espace des règles selon les sources. Par exemple, la partition P2, illustrée dans la Figure 3.14, contient toutes les règles qui concernent VM2. Ces règles (A1, A2, A4) peuvent, par exemple, coexister dans l'un des équipements suivants : HS1 ou ToR1.

Le placement des partitions, résultantes de la phase de division de l'espace, est effectué sur deux étapes :

1. La première étape place les partitions en tenant compte des ressources disponibles. Vu l'existence de plusieurs types de ressources (CPU, memory), les auteurs ont défini la fonction $F(p,\ d)$ pour remédier à ce problème. En effet, cette fonction indique le taux de ressources consommé lorsque la partition p est placée dans l'équipement d. L'algorithme de placement opère comme suit : il commence par choisir, aléatoirement, une partition p et la placer dans un équipement vide d. Ensuite, il ajoute les partitions les plus similaires à p jusqu'à ce que les ressources de d seront épuisées. Calculer la similarité, entre deux partitions, revient à calculer le nombre de règles partagées entre ces partitions. Ceci peut être mis en oeuvre moyennant la fonction $F(P,\ d)$:

$$Similarity(P_i\ ,\ P_k) = F(P_i, d)\ +\ F(P_k, d)\ - F(P_i\ \cup\ P_{k,d}, d)$$

2. La deuxième étape vise à réduire le nombre d'équipements à traverser avant qu'un flux, provenant d'un VM source, soit traité par la règle qui lui correspond dans la politique initiale. Pour ce faire, les auteurs ont proposé un algorithme qui raffine le placement résultant de la première étape en essayant de déplacer les partitions. En effet, cet algorithme essaie de déplacer la partition P_i vers l'équipement le plus proche de la source associée et ayant suffisamment de ressources disponibles. Un tel équipement est désigné par $b(P_i)$. Le choix de la partition P_i à déplacer est basé sur le calcul de la fonction $M(P_i, j)$. Cette fonction désigne le profit résultant du déplacement de la partition P_i vers l'équipement j. Ce profit inclut à la fois le gain acquis en rapprochant la partition de la source correspondante et le potentiel du gain qui peut être engendré par l'espace libéré après le déplacement de P_i.

Ces trois articles permettent de placer les règles dans des environnements hétérogènes. Un tel environnement contient des commutateurs qui supportent l'installation des règles et d'autres équipements divers. Notre outil peut s'adapter afin de supporter ce genre d'environnements.

Tout comme les deux travaux présentés dans la section précédente, Palette, vCrib et (Wang et al., 2014) ne fonctionnent pas dans le cas où la capacité totale disponible sur les commutateurs ne permet pas d'installer toutes les règles. De plus, Palette exige que tous les

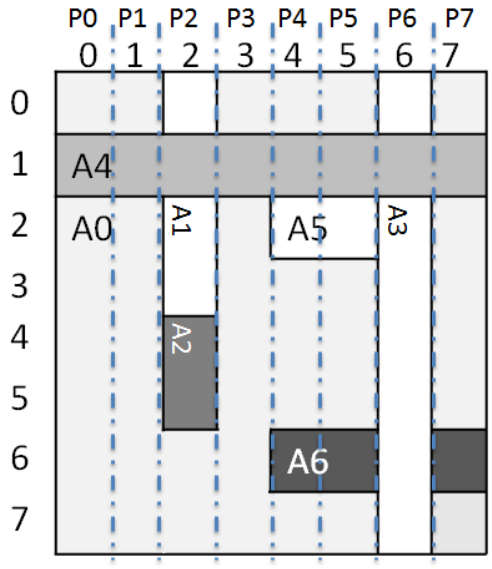


Figure 3.14 Partitionnement d'un espace des entêtes

commutateurs doivent avoir la même capacité. Notre outil surmonte cette exigence en permettant à chaque commutateur d'avoir une capacité différente.

De son côté, le travail de (Wang et al., 2014) ne considère pas les contraintes de capacités des commutateurs.

3.3.2 Placement avec changement de routage

Placement dans des environnements homogènes

Le travail de (Nguyen et al., 2014) défend le fait qu'on peut relaxer le routage afin de trouver un meilleur placement pour les règles, définissant une politique donnée. En effet, ce travail néglige l'impact du changement de routage, tant que le comportement initial est gardé et que la politique est renforcée. La relaxation de routage permet d'exploiter les chemins qui peuvent supporter plus de règles. Les auteurs ont défini un modèle qui permet d'abstraire la sémantique des règles. Ils présument que la politique de base est illustrée uniquement par des règles exprimant l'accessibilité. En effet, pour chaque flux, la politique associée indique les points de sortie accessibles par les paquets de ce flux. À titre d'exemple, ce modèle d'abstraction est illustré dans la Figure 3.15. Étant donné le flux $R1$, sa politique indique que les paquets associés doivent être acheminés vers le point de sortie Exit(E). Au début, les commutateurs contenaient juste les règles 'Default' qui indiquent le chemin par défaut de

chaque flux. Dans cet exemple, le chemin par défaut mène vers le contrôleur. Renforcer la politique d'accessibilité relative à $R1$ revient à installer deux règles ($R1 \rightarrow D, R1 \rightarrow E$) dans les commutateurs C et D. Ces règles d'acheminement permettent d'expédier les paquets du flux $R1$ vers le point de sortie E.

Exprimer des politiques, autres que l'accessibilité, peut être mise en oeuvre en se basant sur le modèle qu'on vient de décrire. Nous allons, tout d'abord, illustrer la solution qui permet de placer les règles renforçant l'accessibilité. Ensuite, nous allons expliquer l'expression d'autres politiques de gestion telle que l'ACL, moyennant l'accessibilité.

Le problème de placement est résolu moyennant un programme linéaire en nombres entiers (ILP). Cet ILP a pour but de maximiser le placement des règles tout en respectant les contraintes suivantes :

1. Chaque commutateur a une capacité limitée.
2. Chaque point de sortie du réseau a une capacité limitée. Ceci implique que la quantité du trafic acheminée vers ce point ne doit pas dépasser une quantité bien déterminée. Le reste du trafic sera acheminé vers le contrôleur.

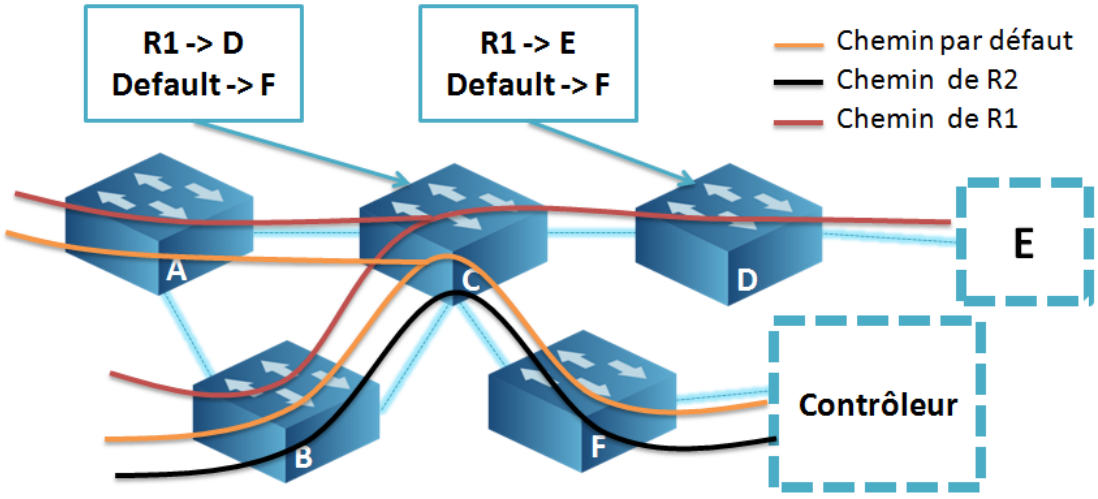


Figure 3.15 Exemple de placement avec relaxation

Considérons les définitions suivantes :

1. PR : dénote l'ensemble des domaines d'application. Chaque domaine $pr \in PR$ contient l'ensemble des paquets concernés par une règle r.

2. $Eg(pr)$: dénote l'ensemble des points de sortie vers lesquels on peut acheminer les flux correspondant au modèle pr.
3. $flow$: dénote l'ensemble des flux qui correspondent à un modèle.
4. $iFlow$: dénote le sous-ensemble de $flow$ qui émerge d'un même point d'entrée .
5. F : dénote l'ensemble des $iFlow$.
6. p_{rate} : dénote la fréquence d'entrée des paquets appartenant à un $iFlow$ $f \in F$.
7. $w_{f,eg}$: dénote la valeur ajoutée par l'acheminement d'un $iFlow$ $f \in F$ à travers un point de sortie $eg \in Eg(pr)$.
8. $Successor(f, eg, s)$: cette fonction définie le chemin suivi par le $iFlow$ f pour atteindre le point de sortie eg quand le commutateur s contient une règle qui concerne f.
9. $DefPath(f)$: cette fonction définie le chemin suivi, par défaut, par le $iFlow$ f s'il n'y a aucune règle qui correspond à ce $iFlow$.

L'ILP définit deux variables booléennes $y_{f,eg}$ et $a_{pr,eg,s}$: $y_{f,eg}$ indique si le $iFlow$ f sera délivré au point de sortie eg. $a_{pr,eg,s}$ indique si on a une règle installée dans s qui permet d'acheminer les flux correspondants à pr vers le point de sortie eg.

La fonction objectif de cet ILP vise à maximiser la valeur obtenue en terme de $iFlow$ acheminés. Ceci revient à maximiser la satisfaction des $iFlow$ dont l'acheminement possède une plus grande valeur ajoutée.

$$max \sum_{f \in F} \sum_{eg \in Eg(pr_f)} w_{f,eg} y_{f,eg}$$

L'ILP définit plusieurs contraintes, parmi lesquelles on peut citer :

1. $\forall f \in F : \sum_{eg \in Eg(pr_f)} y_{f,eg} \leq 1$: cette contrainte indique que les paquets relatifs à f doivent être acheminés vers un seul point de sortie qui appartient à leur ensemble $Eg(pr_f)$.
2. $\forall f \in F, \forall eg \in Eg(pr_f) : y_{f,eg} \leq \sum_{s \in DefPath(f)} a_{pr_f,e,s}$: cette contrainte indique que si f est acheminé vers l'un de ces points de sorties alors on doit avoir une règle installée dans l'un des commutateurs du chemin par défaut de f afin de dévier ce flux.
3. $\forall f \in F, \forall eg \in Eg(pr_f), \forall s \in DefPath(f) : y_{f,eg} \geq a_{pr_f,e,s}$: cette contrainte indique que si f n'est pas délivré à l'un de ses points de sorties alors il n'y aurait aucune règle correspondante installée dans l'un des commutateurs de son chemin par défaut.

4. $\forall\ pr\ \in PR,\ \forall\ s\ \in S\ : \sum_{eg \in Eg(pr)}\ a_{pr,eg,s} \leq 1$: cette contrainte impose indique que chaque modèle de correspondance doit avoir un maximum d'une règle par commutateur pour qu'il soit acheminé à l'un de ces points de sortie.
5. $\forall\ f\ \in F,\ \forall\ eg\ \in Eg(pr_f),\ \forall\ s\ \in DefPath(f), \forall\ n\ \in Successor(f, eg, s)$: $(a_{pr_f,eg,s} - a_{pr_f,eg,n})\Delta(f, eg, n) \leq 0$: Cette contrainte permet d'éviter les boucles d'acheminement. Pour ce faire, elle impose que l'installation d'une règle d'accessibilité d'un flux f, dans un commutateur s doit être suivi par l'installation d'une règle d'accessibilité dans chaque commutateur suivant qui mène vers le point de sortie correspondant, à l'exception des commutateurs qui contiennent une règle par défaut qui achemine f. $\Delta(f, eg, n)$ est égale à 0 si on a une règle par défaut et à 1 sinon.
6. $\forall\ s \in S : \sum_{pr \in PR} \sum_{eg \in Eg(pr)} a_{pr,eg,s} \leq cap(s)$: cette contrainte indique que la somme des règles installées sur un commutateur s ne doit pas dépasser sa capacité $cap(s)$.

L'expression d'une politique ACL peut être faite moyennant le modèle précédent. Pour ce faire, on présume que le chemin par défaut de chaque flux mène, désormais, vers le point de sortie correspondant et que l'action de chaque règle installée est égale à "SUPPRIMER". On exprime juste la suppression vu que l'acceptation est déjà assurée par l'accessibilité. La suppression d'un flux donné peut être mise en oeuvre moyennant l'affectation suivante :

$$Successor(f, eg, s) = s$$

Cette affectation aura un impact sur la contrainte d'évitement de boucles. En effet, cette contrainte exprimera, désormais, que seule une règle sera installée dans le chemin menant vers le point de sortie relatif à f. Cette règle permettra la suppression de ce flux.

Cet article considère le cas où la capacité totale des commutateurs ne sera pas suffisante pour installer toutes les règles. Cependant, il ne traite que les règles qui se basent sur la notion de l'accessibilité. Notre outil permettra de placer des règles exprimées moyennant un espace des entêtes, indépendamment de leurs sémantiques. Ceci appuie le fait que notre travail soit plus générique. De plus, le travail de (Nguyen et al., 2014) ne peut pas installer des règles en ligne vu qu'il se base sur un ILP. Nous proposons, par contre, un algorithme glouton qui sera capable de placer les règles en un temps négligeable.

Dans d'autres mesures, ce travail semble être le seul qui relaxe le routage afin d'augmenter la capacité disponible pour l'installation des règles. Comme dans tous les autres travaux de placement, présentés tout au long de cette revue, le routage est défini pour répondre à plusieurs autres contraintes (bande passante, points d'acheminement ...) qui vont au-delà de placement des règles. La relaxation d'un tel routage doit, par conséquent, conserver toutes

les contraintes considérées lors de son établissement.

Placement dans des environnements hétérogènes

L'outil DIFANE, développé dans le cadre de l'article (Yu et al., 2011), est l'un des premiers outils qui permettent de décharger les contrôleurs du traitement de chaque flux rentrants dans le réseau en installant les règles correspondantes dans le plan de données. Cependant, DIFANE considère que le réseau est formé de deux types de commutateurs qui sont, en l'occurrence, les commutateurs autorités 'authority switch', et les commutateurs ordinaires.

Étant donné un espace d'entêtes mettant en oeuvre un ensemble de règles, DIFANE divise cet espace en des partitions qui seront affectées aux commutateurs autorités. Une partition peut être affectée à plusieurs commutateurs autorités à la fois. Les commutateurs autorités auront une plus grande capacité de règles.

DIFANE traite les flux z_i, rentrant dans le réseau, de la façon suivante. Le commutateur, représentant le point d'entrée d'un nouveau flux, vérifie s'il y a une règle qui correspond à ce flux. Dans le cas contraire, ce commutateur achemine le premier paquet de ce flux vers le commutateur autorité contenant les règles adéquates. Le commutateur autorité traite ce paquet et installe la règle r_i qui y correspond, à l'aide du contrôleur, dans le commutateur d'entrée. Ceci permet au reste des paquets de ce flux d'être traités localement dans le commutateur d'entrée. Si r_i ne possède pas la plus haute priorité alors les autres règles correspondantes à z_i, devraient être installées dans le commutateur d'entrée pour éviter toute incohérence. Les règles, nouvellement installées dans le commutateur d'entrée, seront supprimées automatiquement si aucun flux entrant au réseau ne leur correspond pendant une période de temps.

Comme le montre le Tableau 3.4, les domaines des règles de départ sont exprimées moyennant des intervalles (dimensions) qui peuvent être des singletons. Par exemple, le champ de correspondance F_2 de la règle R_1 doit être dans l'intervalle [14-15] alors que F_5 doit être égale à l'intervalle singleton [0, 0]. Les règles de l'exemple sont définies sur 5 dimensions ($F_1 ... F_5$).

Étant donnée un ensemble de règles définies sur K dimensions, DIFANE divise l'espace, représentant ces règles, en m hypercubes. Chaque hypercube sera placé dans un commutateur autorité. Cette division doit minimiser le nombre de règles, qui en résultent, pour consommer moins d'espace dans les commutateurs autorités. Pour ce faire, les auteurs ont proposé un algorithme de division qui se base sur un arbre de décision dont chaque noeud représente un hypercube. La racine de l'arbre représente tout l'espace des règles. À chaque itération de l'algorithme de division, on choisit le noeud (hypercube) contenant un nombre de règles

qui excède la capacité de chaque commutateur autorité. Cet hypercube sera divisé en des sous-hypercubes qui seront représentés par de nouveaux noeuds dans l'arbre de décision. La division en sous-hypercubes est faite selon une dimension i. Par exemple, la Figure 3.16 montre que la racine, représentant tout l'espace des règles du Tableau 3.4, est divisée selon la dimension F_2. Cette division a créé 3 sous-hypercubes. Chaque sous-hypercube correspond à un intervalle de F_2.

Tableau 3.4 Une liste de règles représentées moyennant des intervalles

Règle \Champ	F1	F2	F3	F4	F5	Action
R1	0-1	14-15	2	0-3	0	accepter
R2	0-1	14-15	1	2	0	accepter
R3	0-1	8-11	0-3	2	1	supprimer
R4	0-1	8-11	2	3	1	supprimer
R5	0-15	0-7	0-3	1	0	accepter
R6	0-15	14-15	2	1	0	accepter
R7	0-15	14-15	2	2	0	accepter
R8	0-15	0-15	0-3	0-3	0-1	supprimer

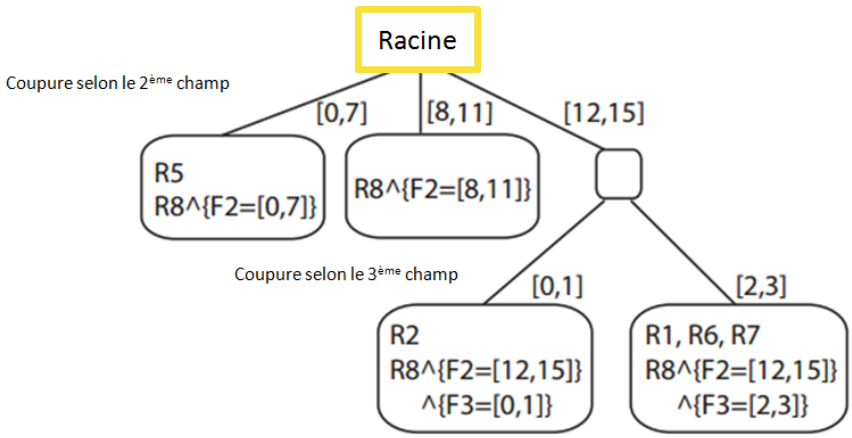


Figure 3.16 Exemple de division des domaines des règles

On choisit toujours la dimension i ayant le maximum de règles ne présentant pas de chevauchements. La sélection des intervalles de divisions (exp : [0, 7] ; [8, 11] et [12, 15] dans la première division de l'exemple précédent) essaie de minimiser l'ensemble des règles résultantes dans les sous-hypercubes. L'algorithme s'arrête quand le nombre de règles, contenues dans chaque feuille de l'arbre de décision, sera inférieur ou égal à la capacité d'un commutateur autorité.

DIFANE place toutes les règles dans les commutateurs autorités. Cette solution est gourmande en ressources vu qu'elle impose à ces commutateurs de posséder de très grandes capacités. Ceci pose un problème, étant donné que la possession de tels commutateurs peut être très coûteuse.

3.4 Divers

3.4.1 Gestion du trafic

Tout au long de cette section, nous allons présenter les travaux qui visent, généralement, à calculer le routage dans le cadre de SDN. Parmi ces travaux, on peut citer celui de (Agarwal et al., 2013), élaboré dans les laboratoires de Bell et Alcatel-Lucent. Ce travail a pour but de calculer le routage dans le cadre des réseaux contenant à la fois des commutateurs SDN et des commutateurs ordinaires. Ce cas peut se produire quand on entame un processus de migration incrémentale vers SDN. Cependant, l'introduction des commutateurs SDN doit être accompagnée par une reconfiguration de ces commutateurs afin de conserver la politique de routage initiale représentée en partie dans les commutateurs ordinaires. L'ancienne politique de routage est exprimée moyennant le protocole OSPF qui se base sur la notion de plus court chemin pour trouver les chemins adéquats.

Considérons les définitions suivantes :

1. C : dénote le sous-ensemble de S représentant les commutateurs compatibles SDN.
2. O : dénote le sous-ensemble de S représentant les commutateurs ordinaires.
3. $\forall\ e\ \in E : c(e)$: dénote la capacité par lien utilisé dans le protocole OSPF.
4. $\forall\ s, d\ \in S : T_{s,d}$: représente le taux de trafic entre s et d.
5. $\forall\ s, d\ \in S : P_{s,d}$: désigne l'ensemble des chemins entre s et d.
6. Un trafic dont le routage est défini moyennant OSPF et qui passe par un commutateur SDN avant d'atteindre sa destination, est appelé trafic contrôlable.
7. $\forall\ u\ \in C, \forall\ d\ \in S : I_{u,d}$: dénote le trafic injecté par u et allant vers d. Ce trafic peut avoir une source différente de u.
8. $\forall\ e\ \in E : g(e)$: indique le trafic incontrôlable qui passe par le lien e.

Conserver la même politique de routage, après avoir substitué quelques commutateurs ordinaires, revient à calculer les chemins des trafics contrôlables en utilisant le minimum des

liens dans le réseau. Minimiser le nombre de liens à utiliser revient à minimiser la perte des paquets ainsi que la latence dans chaque lien. Le calcul de ces chemins est fait moyennant un programme linéaire qui a $z(P)$ comme variable. Cette variable dénote le flux contrôlable acheminé à travers le chemin P. La variable θ exprime le taux maximum d'utilisation des liens.

Les contraintes du programme linéaire sont définies comme suit.

1. $\forall\ e \in E : g(e)\ + \sum_{P:P\ni e} z(P)\ \leq \theta\ c(e)$: cette contrainte exprime que le flux total acheminé à travers un lien e ne doit pas dépasser la capacité qui lui est réservée. Le flux total est exprimé comme la somme de flux incontrôlable et des flux contrôlables.
2. $\forall\ u \in C,\ \forall\ d \in N : \sum_{P\in P_{ud}} z(P)\ \geq I_{ud}$: cette contrainte indique que tout flux injecté doit être acheminé.

L'objectif de ce programme revient à minimiser θ. En conséquence, si θ aura une valeur optimale qui soit inférieur à 1, alors il n'y aura aucun lien $e$qui sera surexploité.

Outre le calcul de routage, cet article traite aussi le problème de placement des commutateurs SDN. Étant donné un ensemble de commutateurs SDN et une matrice $\{T_{s,d}\}_{s,d\ \in\ N}$ exprimant le trafic entre chaque paire de noeuds du réseau, le placement vise à maximiser le flux qui sera acheminé à travers les chemins contrôlables.

Le travail de (Giroire et al., 2014) vise principalement à calculer le routage dans des réseaux composés exclusivement de commutateurs SDN, tout en économisant de l'énergie. L'économie de l'énergie dans un réseau peut être mise en oeuvre en minimisant le nombre de liens utilisés pour implanter la politique de routage. Le calcul de routage est réalisé moyennant un programme linéaire en nombres entiers. Outre l'économie de l'énergie, l'ILP prend en compte les différentes capacités des commutateurs. En conséquence, le nombre de règles installées dans un commutateur s, pour mettre en oeuvre la politique de routage, ne doit pas dépasser $cap(s)$.

Considérons les définitions suivantes :

1. D : est l'ensemble des flux à acheminer.
2. $\forall\ s,t\ \in N : D^{s,t}$: représente le flux à acheminer entre s et t.
3. $\forall\ n\ \in N\ :\ adj(n)$: dénote l'ensemble des commutateurs adjacents à n.
4. $\forall\ u,v \in N : a_{u,v}$: est une variable binaire qui indique si le lien (u,v) est actif ou non.
5. $\forall\ s,t,u,v \in N,\ f^{s,t}_{u,v}$: est une variable qui indique si le flux allant de s vers t est acheminé à travers $(u,\ v)$ moyennant une règle, autre que celle par défaut.

6. $\forall\ s, t, u, v \in N,\ g_{u,v}^{s,t}$: est une variable qui indique si le flux allant de s vers t est acheminé à travers $(u,\ v)$ moyennant une règle par défaut.

L'ILP définit plusieurs contraintes, parmi lesquelles on peut citer :

1. $$\forall\ u \in N, \forall\ D^{s,t} \in D : \sum_{v \in\ adj(u)} (\ f_{v,u}^{s,t} + g_{v,u}^{s,t} - g_{u,v}^{s,t} - f_{u,v}^{s,t}) = \begin{cases} 1 & \text{si } u = s \\ -1 & \text{si } u = t \\ 0 & \text{sinon} \end{cases}$$

   Cette contrainte indique que les flux rentrant dans un commutateur doivent être égaux aux flux qui y sortent, à l'exception de la source s et de la destination t.

2. $\forall\ (u, v)\ \in\ E\ :\ \sum_{D^{s,\ t} \in D} D^{s,t}(\ f_{v,u}^{s,t} + g_{v,u}^{s,t} + g_{u,v}^{s,t} + f_{u,v}^{s,t}) \leq\ \mu\ cap(u, v)\ a_{u,\ v}$: cette contrainte indique que la totalité des flux acheminés à travers un lien (u,v) ne doivent pas excéder sa capacité. μ indique le taux maximum qui peut être utilisé dans un lien.
3. $\forall\ u\ \in N : \sum_{D^{s,t} \in D}\ \sum_{v \in adj(u)} f_{u,v}^{s,t} \leq cap(u) - 1$: cette contrainte indique que la totalité des règles installées dans un commutateur afin de définir la logique de routage ne doit pas dépasser sa capacité. Mentionnant que la règle par défaut consomme déjà une unité de capacité dans le commutateur.

La fonction objectif de l'ILP vise à minimiser le nombre de liens utilisés dans le routage moyennant l'expression suivante :

$$min \sum_{(u,v) \in E} a_{u,v}$$

Étant donné que cet ILP est NP-difficile, les auteurs de l'article ont défini un algorithme glouton afin de trouver le routage qui économise l'énergie en un temps raisonnable. L'algorithme commence à rechercher, pour chaque flux $D^{s,t}$, le plus court chemin qui respecte les contraintes sur les capacités des liens et des commutateurs. Après cela, on diminue la capacité consommée dans chaque commutateur, en regroupant les règles qui acheminent le plus de flux vers le même port, dans la règle par défaut. Par exemple, le premier tableau de la Figure 3.17 représente un ensemble de règles qui définissent pour chaque flux (s,t), le port à travers lequel il sera acheminé. Étant donné que Port-3 est utilisé dans trois règles parmi quatre, alors on met l'action d'acheminement vers ce port comme action par défaut. Ceci diminue le nombre de règles, comme le montre le 2e tableau de la Figure 3.17.

Les deux travaux présentés ci-dessus permettent de calculer le routage dans le cadre de SDN.

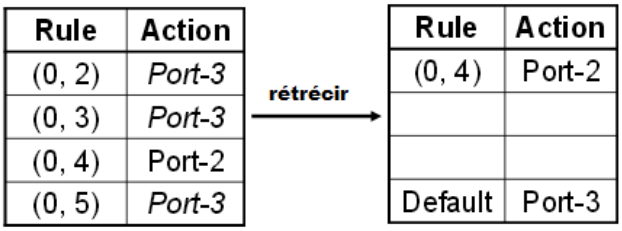


Figure 3.17 Diminution de la capacité consommée

Le travail de (Agarwal et al., 2013) opère dans des environnements hétérogènes tandis que le travail (Giroire et al., 2014) traite le cas des réseaux homogènes. On remarque que ces travaux ainsi que ceux présentés dans la section (3.2.2) permettent de respecter la capacité imposée à chaque lien $e \in E$. Cependant, les travaux présentés dans cette section ne permettent ni la garantie ni la limitation de la bande passante.

3.4.2 Renforcement à partir des scénarios

L'article de (Yuan et al., 2014) se situe dans le cadre du renforcement de politiques de gestion des réseaux SDN. Cependant, la politique de gestion est exprimée sous forme de scénarios pour faciliter la tâche aux administrateurs qui ne maîtrisent pas la programmation. À titre d'exemple, la Figure 3.18 illustre une politique de sécurité exprimée à l'aide de trois scénarios. Cette politique exprime la suppression et l'acceptation de quelques flux. L'exemple considéré identifie un flux en utilisant le 3-uplet <inPort, srcIP,dstIP>. Le champ inPort désigne le port d'entrée de flux tandis que srcIP et dstIP désignent respectivement l'adresse de la source et celle de la destination de flux.

Le premier scénario de la Figure 3.18 indique que tout flux qui arrive sur le port d'entrée 1 sera accepté, indépendamment des adresses de la source et de la destination.

Le 3e scénario indique que les paquets arrivant de ip2 et allant vers ip1 à travers le port 2 seront acceptés s'il y a eu un envoi de paquets de ip1 vers ip2 à travers le port 1.

L'outil résultant de cet article est appelé NetEgg. Cet outil met en oeuvre la politique exprimée moyennant les scénarios en générant un tableau de politique, un programme contrôleur et plusieurs tableaux d'états. Ces derniers permettent de garder les informations nécessaires au renforcement des politiques. Par exemple, dans le 3e scénario on aura besoin d'un moyen pour savoir s'il y a eu d'envoi de paquets de ip1 vers ip2 à travers le port 1. Pour ce faire, on

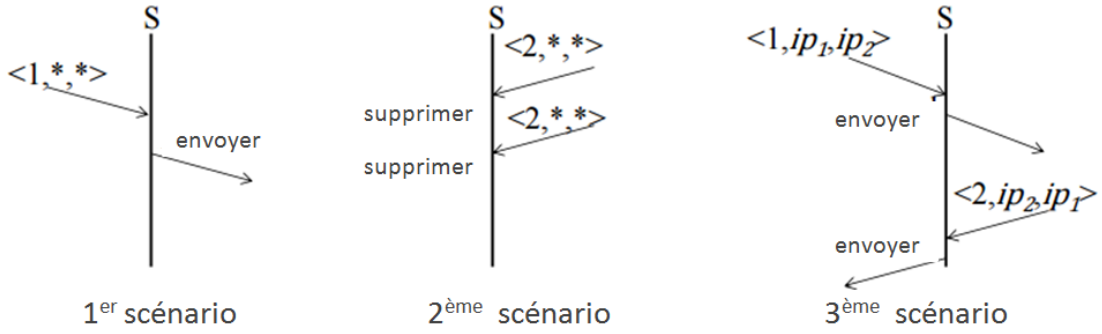


Figure 3.18 Exemples de scénarios

peut utiliser un tableau d'états ST qui associe à ip1 la valeur 0 si ip1 a envoyé des paquets à ip2 et la valeur 1 sinon. Cette valeur sera utilisée par la suite pour déterminer si on doit accepter le flux provenant de ip2 et allant vers ip1 à travers le port 2.

Le tableau de la politique définit, pour chaque flux, l'action qu'il faut exécuter ainsi que les mises à jour qu'il faut effectuer dans les tableaux d'états. Comme le montre le Tableau 3.5, chaque tableau de politique contient 4 champs :

1. Domaine : indique le domaine des paquets.
2. Tests : spécifie les conditions sur les valeurs contenues dans les tableaux d'états.
3. Action : dénote l'action qui doit être exécutée lorsque le paquet, en entrée, vérifie les conditions de Match et Tests.
4. Updates : indique les mises à jour à appliquer sur les tableaux d'états.

Le programme de contrôle traite les paquets qui arrivent au réseau en utilisant les règles installées dans le tableau de la politique.

Tableau 3.5 Un exemple d'un tableau de politique

Domaine	Tests	Action	Mise à jour
port=1	-	envoyer	ST(dstip) :=1
port=2	ST(srcip)=1	envoyer	-
port=2	ST(srcip)=0	supprimer	-

Étant donné n scénarios ($sc_1 ... sc_n$), NetEgg génère le tableau de politique et les tableaux d'états d'une façon itérative en utilisant toutes les combinaisons de correspondances présentes

dans les scénarios . À chaque itération, il vérifie la consistance du tableau de la politique vis-à-vis les scénarios. Malgré le fait qu'il facilite la tâche aux administrateurs, NetEgg utilise un algorithme de génération de contrôleur, de complexité exponentielle en temps. Ceci ne permettra pas de s'adapter rapidement aux changements de politiques. De plus, NetEgg synthétise uniquement le comportement du contrôleur en ignorant la configuration du plan de données. Ce modèle peut alourdir la tâche du contrôleur qui sera obligé d'intercepter tous les flux rentrants dans le réseau afin de mettre en oeuvre la politique de gestion.

3.4.3 Méthodes de vérification des aspects

L'émergence de SDN vise à faciliter la gestion des réseaux en séparant le plan du donnée du plan de contrôle et en définissant des interfaces de programmation unifiées. Toutefois, l'implantation de politiques de gestion demeure encore sujette aux erreurs. Pour remédier à cela, plusieurs travaux ont été élaborés. En effet, ces travaux permettent de vérifier l'exactitude des politiques implémentées en utilisant divers techniques.

L'outil Veriflow, introduit dans le travail de (Khurshid et al., 2012), vérifie, en ligne, la conformité de la configuration du plan de données vis-à-vis de certains invariants qui définissent les politiques de gestion du réseau. En effet, Veriflow intercepte les mises à jour, appliquées par le contrôleur, et vérifie en temps réel si l'ajout, la modification ou la suppression d'une règle peut violer certains invariants du réseau.

Veriflow permet de vérifier trois types d'invariants

1. Accessibilité : Vérifier si un paquet donné peut être livré à sa destination.
2. Absence de boucles d'acheminement.
3. Consistance entre commutateurs : permet de vérifier si deux commutateurs s_1 et s_2 ont le même comportement en ce qui concerne l'acheminement de flux.

Veriflow classifie les paquets circulant dans le réseau en des classes d'équivalences (equivalency classes-ECs). Chaque EC dénote un ensemble de paquets qui subissent la même action d'acheminement dans chaque commutateur $s \in S$. Le comportement de chaque classe d'équivalence EC est modélisé moyennant un graphe d'acheminement. Ce graphe orienté indique la politique d'acheminement de la classe d'équivalence associée. En effet, chaque noeud de ce graphe représente un commutateur du réseau. Chaque arc, allant d'un commutateur s_i vers un commutateur s_j, indique que les paquets de la classe d'équivalence EC seront acheminés vers s_j, lorsqu'ils atteignent s_i.

Veriflow utilise les graphes d'acheminement afin de vérifier les invariants. L'accessibilité d'un paquet est vérifiée en effectuant une recherche en profondeur dans le graphe correspondant à sa classe d'équivalence. Elle sera confirmée si on trouve un chemin qui commence de la source du paquet et qui atteint sa destination. L'absence des boucles d'acheminement dans le réseau correspond à l'absence des cycles dans tous les graphes d'acheminement.

La vérification de la consistance entre deux commutateurs s_1 et s_2 peut être réalisée en parcourant chaque graphe d'acheminement à partir des noeuds correspondants à ces commutateurs et en vérifiant si on a le même comportement d'acheminement pour chaque classe d'équivalence. Pour accélérer le processus de vérification, Veriflow construit juste les graphes de cheminement des classes d'équivalences concernées par les invariants.

Tout en restant dans le même contexte, le travail de (Kazemian et al., 2013) a donné naissance à l'outil de vérification NetPlumber. Cet outil est le fruit d'une collaboration entre l'université de Stanford, Google Inc. et Microsoft Research. Il consiste en un outil de vérification temps réel des invariants, qui se base sur l'analyse de l'espace des entêtes.

Outre l'absence des boucles d'acheminement et l'accessibilité, NetPlumber permet de vérifier d'autres invariants qui sont en l'occurrence :

1. Le passage obligatoire par des points d'acheminement pour certains flux.
2. La limitation de la longueur des chemins d'acheminement.

Pour ce faire, NetPlumber se base sur un graphe appelé, 'plumbing graph'. Ce graphe dirigé est structuré comme suit. Chaque noeud correspond à une règle d'acheminement tandis que chaque arc (r_i, r_j) représente le flux qui résulte de l'application de la règle r_i et qui va subir l'action de la règle r_j. Ces arcs sont appelés tubes (pipes) vu qu'ils servent comme conduites pour les différents flux. Le flux qui peut circuler sur le tube (r_i, r_j) englobe tous les paquets produits suite à l'action de r_i et qui peuvent être traité par r_j.

La construction du 'plumbing graph' prend en compte les priorités entre les règles qui se situent dans le même commutateur. Soit r_i et r_j deux règles installées dans le même commutateur tel que $r_i.priority > r_j.priority$. Le tube liant r_i et r_j dans le 'plumbing graph' peut conduire tous les paquets qui correspondent à la différence entre le domaine de r_i et celui de r_j. Ceci met en oeuvre le fait que les paquets qui ne correspondent pas à r_i seront traités par r_j s'ils appartiennent à son domaine d'application.

À titre d'exemple, la Figure 3.19 présente un 'plumbing graph' généré à partir d'un réseau constitué de quatre commutateurs. Comme le montre la figure, chaque commutateur contient un ensemble de règles. Le 'plumbing graph' lie ces règles entre elles par deux types d'arcs :

1. Les arcs (tubes) noirs font lier les règles situées dans les commutateurs. Le tube liant la 3e règle du tableau 1 avec la 1re règle du tableau 3 accepte les paquets du domaine “101xxxxx”. Ce domaine correspond à l’intersection du domaine des paquets traités par la 3e règle du tableau 1 avec le domaine des paquets concernés par la 1re règle du tableau 3
2. Les arcs rouges mettent en évidence les relations entre les règles localisées dans les mêmes commutateurs. On remarque que la première règle du tableau 2 a une influence sur la 2e règle du même tableau vu que l’intersection de leurs modèles est égale à (match :1011xxxx, in-port :4).

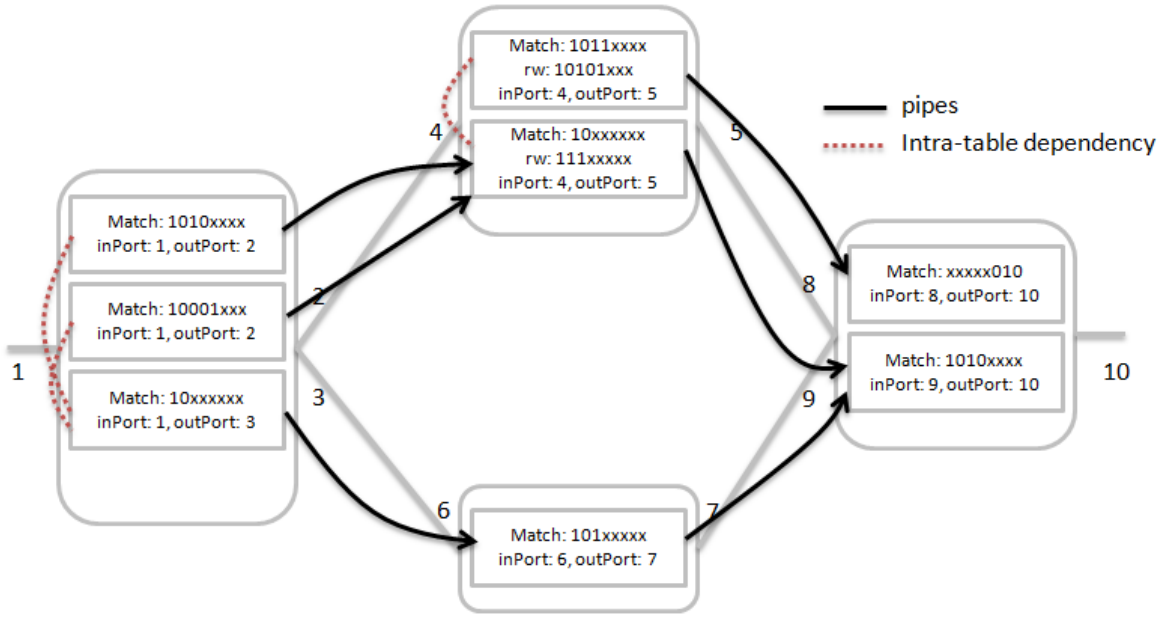


Figure 3.19 Exemple d’un ‘plumbing graph’

Netplumber vérifie les invariants en décorant le ‘plumbing graph’ par d’autres types de noeuds. Parmi ces noeuds, on peut citer les noeuds sources et les noeuds sondes. Un noeud source sert comme un générateur de flux qui peut se mettre à un point donné de réseau pour propager un flux et vérifier sa portée. Un noeud sonde peut intercepter un flux afin de vérifier s’il correspond à une condition donnée.

Considérons l’invariant suivant défini par rapport au réseau de l’exemple de la Figure 3.19 :

1. Les flux provenant du port 1 et allant vers le port 10 doivent correspondre au modèle “xxxxx010”.

Vérifier cet invariant revient à lier un noeud source (S) au port 1 et un noeud sonde (P) au port 10 et configurer P pour qu'il affirme que tous les paquets qui y arrivent correspondent au modèle "xxxxx010". Ceci est illustré dans la Figure 3.20. On remarque que la propagation d'un flux générique "xxxxxxxx" à partir de S permet d'atteindre P avec des flux correspondant au modèle "xxxxx010".

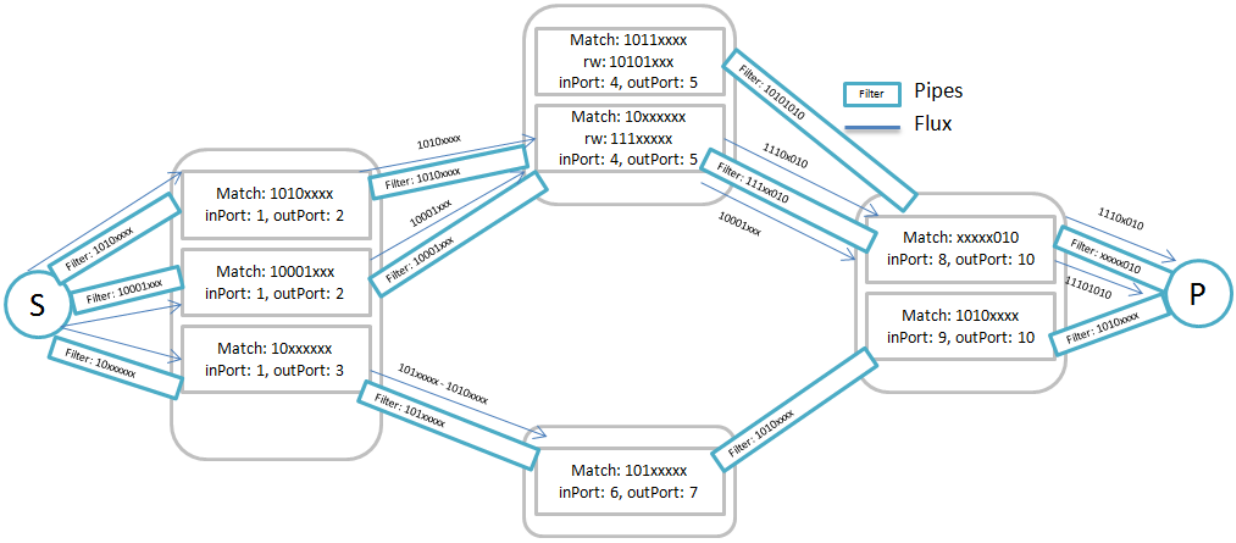


Figure 3.20 Exemple de 'plumbing graph' enrichi par un noeud source et un noeud sonde

La vérification de l'absence des boucles d'acheminement peut se faire en configurant tous les noeuds pour qu'ils vérifient si un flux arrive plus qu'une fois à leurs entrées.

Dans d'autres mesures, la vérification de l'accessibilité exige l'introduction de nouveaux noeuds. Supposons qu'on veut vérifier qu'un serveur S1 ne peut pas être accessible à partir d'un équipement H1. Pour se faire, on lie un noeud source au port d'entrée de H1 et un noeud sonde au port de sortie menant vers S1. Après, on configure le noeud sonde afin de vérifier que tous les flux, qui arrivent y ne proviennent pas de H1.

Les outils qu'on vient de décrire se basent tous sur la construction des graphes qui expriment le comportement des règles déployées dans les commutateurs de réseau. Outre ces outils, on peut citer Anteater qui vérifie les invariants en se basant sur un solveur SAT.

Cet outil, présenté dans le travail de (Mai et al., 2011), permet de vérifier plusieurs invariants tels que l'accessibilité et l'absence de boucles d'acheminement. Pour chaque lien physique $(s_i,\ s_j)$, Anteater associe la fonction $Policy(s_i,\ s_j)$ qui indique le type des paquets qui peuvent traverser ce lien. Par exemple, la fonction suivante affirme que les paquets allant de s_1 vers s_2 doivent avoir l'adresse ip '10.1.2.0' comme source et l'adresse ip '10.1.2.60' comme

destination.

$$Policy(s_1,\ s_2,\ packet) : -packet.srcIp ==\ 10.1.2.0\ \wedge\ packet.dstIp = 10.1.2.60$$

La vérification de l'accessibilité entre deux noeuds se réduit à un problème de décidabilité. Ce problème essaie de trouver un paquet pq et un chemin P qui lient les deux noeuds tel que :

$$\forall (u, v) \in P : Policy(u, v, pq) = true$$

Contrairement à NetPlumber et à Veriflow, Anteater ne convient pas aux tâches de vérification en temps réel vu qu'il se base sur un solveur SAT.

Les techniques utilisées dans ces outils permettent de vérifier une grande variété d'invariants tels que l'accessibilité et les points d'acheminement. Il est donc raisonnable de les utiliser afin de vérifier la rectitude de renforcement de ces invariants, effectué par notre outil.

3.5 Conclusion

Tout au long de ce chapitre, nous avons présenté les différents travaux relatifs à notre sujet. Nous avons, aussi, fait une comparaison entre ces travaux et notre sujet afin de préciser nos contributions. Le chapitre suivant servira à fournir une explication des approches qui mettent en oeuvre nos contributions.

CHAPITRE 4 MISE EN OEUVRE DES ASPECTS

4.1 Introduction

Dans le chapitre précédent, nous avons fait une revue des travaux existants qui traitent principalement le renforcement des aspects considérés par notre travail et de quelques autres aspects connexes tels que la gestion du trafic. Dans ce chapitre, nous présentons les méthodes que nous avons développées afin de mettre en oeuvre les aspects de gestion considérés.

Le premier aspect vise à garantir et à limiter la bande passante pour un flux donné (ex : streaming video). La mise en oeuvre de cet aspect consiste à trouver un routage, entre la source et la destination de flux, qui passe par des liens du réseau qui peuvent assurer cet aspect.

Le deuxième aspect met en oeuvre les politiques de composition des points d'acheminement. Une telle politique impose à un flux donné de passer par des équipements spécifiques du réseau avant d'atteindre leurs destinations. De ce fait, mettre en oeuvre une politique de composition des points d'acheminement pour un flux donné, revient à trouver un routage qui va de sa source de flux pour atteindre sa destination tout en passant par les points d'acheminement spécifiés dans la politique de composition.

Le troisième aspect vise à trouver un placement d'une liste de règles, spécifiant une politique globale du réseau, dans les commutateurs de ce réseau tout en respectant leurs capacités .

La suite de ce chapitre est composée de trois sections. La section 4.2 fournit une définition abstraite d'un réseau ainsi que ces paramètres considérés dans ce mémoire. La section 4.3 présente la méthodologie qui permet de mettre en oeuvre les deux premiers aspects.

Dans la section 4.4, nous présentons deux approches pour résoudre le problème de placement des règles.

4.2 Définitions

Les définitions qui suivent vont être utilisées tout au long de ce chapitre.

Un réseau $N =< D, P, E >$ avec $D = S \cup H$ un ensemble d'équipements, P un ensemble de ports et $E : D \times P \mapsto D \times P$ une relation qui modélise les liens du réseau. S dénote l'ensemble des commutateurs du réseau et H est l'ensemble d'hôtes. $\forall p \in P$ et $\forall d \in D$, p est un port

d'entrée de d si $E^{-1}(d,p) \neq \emptyset$ et p est un port de sortie de d si $E(d,p)$ existe. Les éléments de l'ensemble H sont connectés au reste du réseau à travers des ports d'entrée et de sortie dénotés respectivement par P_{ingres} et P_{egress} et qui sont inclus dans P. Chaque commutateur $s \in S$ a une capacité dénotée par $cap(s)$. Cette capacité indique le nombre maximal de règles qui peuvent être installées dans s.

Chaque lien e a une capacité $cap(e)$ qui dénote le maximum de trafic qui peut circuler sur ce lien. Cette capacité correspond à la bande passante offerte par e.

Pour une source src et une destination dst dans H, la fonction $routingSwitches(src,\ dst)$, retourne la séquence des commutateurs $(s_1...s_k)$ alors que la fonction $routingEdges(src,\ dst)$ retourne la séquence des liens $(e_1...e_k)$ qui relie la source à la destination.

La Figure 4.1 présente un exemple de réseau contenant 12 équipements, dont 5 commutateurs indiqués par $(s_1, ...,\ s_5)$ et 7 hôtes indiqués par $(h_{11},\ h_{12},\ h_{21},\ h_{31},\ h_{41},\ h_{42},\ h_{51})$. Chaque hôte est annoté par son adresse mise entre parenthèses. Par exemple, l'adresse de l'hôte h_{11} est '000'.

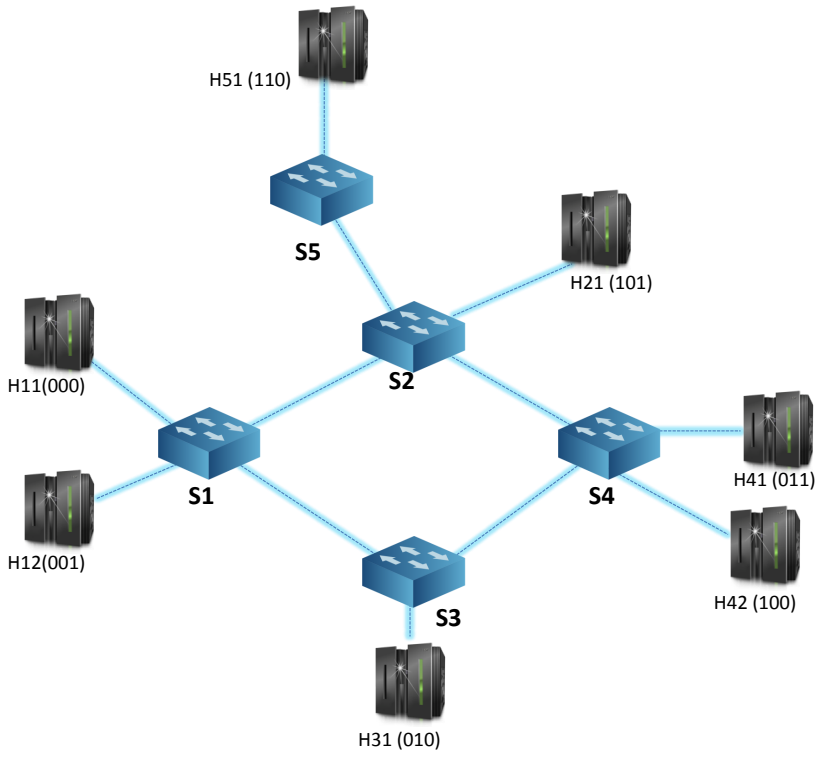


Figure 4.1 Un exemple de réseau

4.3 Aspects de gestion mises en oeuvre moyennant le routage

Étant donné un réseau N et un ensemble de requis qui expriment la bande passante et la politique de composition des points d'acheminement désirés pour chaque flux, le but de cette partie est de mettre en oeuvre ces requis dans le réseau.

La liste suivante présente un exemple de requis qui concerne le réseau de la figure 4-1. Chaque requis est formé par un domaine d'application et des propriétés qui doivent être respectées par les paquets du domaine. L'ensemble des paquets appartenant à un domaine d'application donné forme un flux.

1- (src=100, dst=110, srcPort=80) : max(25 MB/S), min(10 MB/S)

2- (src=000, dst=011, srcPort=80) : waypoint(S2|S3, S4)

Le premier requis illustre l'aspect de la garantie et la limitation de la bande passante. Il définit les contraintes sur la bande passante pour les paquets du domaine (src=100, dst=110, srcPort=80). Ce domaine englobe tous paquets issus de l'hôte '100' via le port 80 et à destination de l'hôte '110'. La bande passante qui concerne ces paquets doit être limitée à 25 MB/S. D'autre part, cette même bande passante doit être garantie à 10 MB/S.

Le deuxième requis illustre l'aspect des points d'acheminement. Il définit une politique de composition de ces points pour les paquets du domaine (src=000, dst=011, srcPort=80). En effet, ces paquets doivent passer par le commutateur S2 ou S3 suivis de S4 avant d'atteindre leur destination.

Le renforcement de ces aspects illustrés par les requis susmentionnés revient à trouver les deux routages suivants :

1. Un routage des paquets du domaine (src=100, dst=110, srcPort=80) qui garantit et limite la bande passante.
2. Un routage des paquets du domaine (src=000, dst=011, srcPort=80) qui respecte la politique de composition en passant par S2 ou S3 suivie de S4.

De ce fait, le renforcement des aspects de la bande passante et des points d'acheminement se réduit à un problème de routage.

Mise à part les requis qui doivent être satisfaits, nous supposons qu'un ancien routage existe et que le calcul d'un nouveau routage doit se rapprocher de l'ancien routage afin de minimiser la reconfiguration du réseau. De plus, mettre en oeuvre un routage revient à installer des règles d'acheminement dans les commutateurs du réseau. Pour l'exemple de la figure 4-1, si

un paquet du domaine (src=000, dst=110) se trouve dans le commutateur s_2 alors on doit avoir une règle d'acheminement installée dans s_2 qui dirige ces paquets vers le commutateur s_5 afin d'atteindre leur destination '110'.

Le calcul d'un nouveau routage doit prendre en compte la limitation de la capacité des commutateurs en terme de règles supportées. De ce fait, le nombre de flux acheminés à travers un commutateur s ne doit pas dépasser la capacité de ce commutateur, exprimée par $cap(s)$.

Le renforcement se base principalement sur un programme linéaire en nombres entiers (ILP). Ce programme vise principalement à trouver un routage qui permet d'acheminer les paquets, correspondant à chaque domaine, de leurs sources vers leurs destinations tout en respectant les contraintes sur la bande passante et sur les points d'acheminement.

Le programme ILP proposé se base sur les notations et définitions suivantes :

1. PR dénote l'ensemble des domaines d'application des requis. Le domaine d'application d'un requis désigne les paquets concernés par ce requis. Chaque domaine pr est défini sous forme de conditions sur les champs des entêtes des paquets. Tout au long de cette section, nous supposons que chaque pr est défini pour une seule source $pr.src$ et une seule destination $pr.dst$. Pour considérer des domaines de correspondance génériques, il suffit d'intégrer une étape préliminaire qui permet de diviser chaque domaine générique en des domaines spécifiques. Chaque domaine spécifique correspondra à une seule source et une seule destination. Une telle étape sera présentée et utilisée dans la section suivante consacrée au problème de placement de règles. Par exemple, le domaine d'application du premier requis est $pr = (src = 100,\ dst = 110,\ srcPort = 80)$.
2. $\forall\ d\ \in D:\ OUT(d)$ dénote le sous ensemble de E contenant les arcs qui sont issus de l'équipement d. Un arc e, défini par $(d_1,\ d_2,\ p_1,\ p_2)$ appartient à $OUT(d)$ si $d = d_1$ et $p_1 \in d_1.outPorts$.
3. $\forall\ d\ \in D:\ IN(d)$ dénote le sous ensemble de E contenant les arcs rentrants de l'équipement d. Un arc e, défini par $(d_1,\ d_2,\ p_1,\ p_2)$ appartient à $IN(d)$ si $d = d_2$ et $p_2 \in d_2.inPorts$.
4. $\forall\ pr\ \in\ PR:\ g_{pr}$ dénote la bande passante qui doit être garantie pour les paquets du domaine pr.
5. $\forall\ pr\ \in\ PR:\ ul_{pr}$ dénote la bande passante limite qui ne doit pas être dépassée par les paquets du domaine pr.
6. $\forall\ e\ \in\ E:\ G^e_{max}$ dénote la proportion maximale réservée à la garantie de la bande passante sur un arc e. Ces valeurs sont définies par les administrateurs du réseau. Elles

expriment le pourcentage de la capacité d'un arc qu'ils désirent réserver à la garantie de la bande passante. Par exemple, si sur un arc e_1 on veut réserver seulement 50% pour la garantie de la bande passante, alors on aura $G_{max}^{e_1} = 0.5$.

7. $\forall\, e \in E :$ UL_{max}^{e} dénote la proportion maximale de la bande passante qui ne doit pas être dépassée sur l'arc e. Ces valeurs sont définies par les administrateurs du réseau. Elles expriment le pourcentage de la capacité d'un arc qu'ils désirent désigner comme limite maximale à ne pas dépasser par n'importe quel flux. Par exemple, si sur un arc e_1 on veut fixer la limite à 90% de la capacité de cet arc, alors on aura $UL_{max}^{e_1} = 0.9$.

À titre d'exemple, le requis suivant exprime la bande passante désirée pour les paquets du domaine pr= (src=000, dst=110). La bande passante à garantir est g_{pr} = 10MB/S et la bande passante limite est ul_{pr} = 25MB/S.

src=000, dst=110 : max(25 MB/S), min(10 MB/S)

La variable principale de notre ILP est dénotée par x_{pr}^{e}. $\forall\, pr \in PR,\ \forall\, e \in E,\ x_{pr}^{e}$ est une variable de décision binaire qui indique l'acheminement des paquets correspondants au domaine pr à travers le lien e. $x_{pr}^{e} = 1$ si les paquets correspondants à pr sont acheminés à travers e. Dans le cas contraire, on aura $x_{pr}^{e} = 0$.

Les contraintes de notre ILP :

$$\forall\, pr \in PR,\ \forall\, d \in D : \sum_{e \in OUT(d)} x_{pr}^{e} - \sum_{e \in IN(d)} x_{pr}^{e} = \begin{cases} 0 \text{ si } d \in S, d \neq pr.src \text{ et } d \neq pr.dst \\ 1 \text{ si } d = pr.src \\ -1 \text{ si } d = pr.dst \end{cases} \tag{4.1}$$

$$\forall\, s \in S,\ \sum_{e \in OUT(s)} \sum_{pr \in PR} x_{pr}^{e} \leq cap(s) \tag{4.2}$$

$$\forall\, e \in E : \sum_{pr \in PR} g_{pr}\ x_{pr}^{e} \leq G_{max}^{e}\ cap(e) \tag{4.3}$$

$$\forall\, pr \in PR\ , \forall\, e \in E :\ ul_{pr} x_{pr}^{e} \leq UL_{max}^{e}\ cap(e) \tag{4.4}$$

La contrainte 4.1 sert à définir un chemin unique entre la source et la destination qui correspondent à chaque domaine pr. Cette contrainte indique que dans chaque commutateur, différent de la source ou de la destination de flux caractérisé par un domaine pr, le nombre d'arcs sortants utilisés pour acheminer ce flux doit être égal au nombre d'arcs rentrants utilisés pour le même but. En d'autres termes, chaque flux qui rentre dans un commutateur par un arc d'entrée doit sortir par un arc de sortie si ce commutateur est différent de la source et

de la destination de ce flux. Au niveau de la source, on utilise seulement un arc sortant pour acheminer le flux ce qui explique que la différence est égale à 1. Au niveau de la destination, on utilise un seul arc rentrant pour acheminer le flux à sa destination d'où la différence est égale à -1.

La contrainte 4.2 vise à respecter la capacité réservée aux règles de routage sur chaque commutateur. En effet, chaque décision de routage $x_{pr}^{e} = 1$ est mise en oeuvre moyennant une règle d'acheminement qui sera installée dans le commutateur s tel que $e \in OUT(s)$. De ce fait, la somme des $x_{pr}^{e} = 1$ pour tous les arcs e issus d'un commutateur s ne doit pas dépasser la capacité de ce commutateur.

Les contraintes 4.3 et 4.4 servent, respectivement, à la garantie et à la limitation de la bande passante. La contrainte 4.3 garantit la bande passante en imposant que la somme des quantités de bande passante à garantir sur chaque arc e ne doit pas dépasser la capacité réservée à la garantie de la bande passante au sein de cet arc. La capacité réservée à la garantie de la bande passante au sein d'un arc e est égale à $G_{max}^{e}\ cap(e)$. La contrainte 4.4 indique que si un arc e est réservé à l'acheminement des paquets correspondant à un domaine pr, alors on doit avoir la possibilité d'atteindre la bande passante limite ul_{pr} sur cet arc. De ce fait, ul_{pr} doit être inférieure ou égale à la bande passante limite maximum dénotée par $UL_{max}^{e}\ cap(e)$.

L'objectif de notre ILP est de maximiser l'expression suivante :

$$\sum_{e \in E} \sum_{pr \in PR} coef_{pr}^{e} x_{pr}^{e} \tag{4.5}$$

$$t.q.\ coef_{pr}^{e} = \begin{cases} 1 & \text{si } e \in routingEdges(pr.src, pr.dst) \\ -1 & \text{sinon} \end{cases}$$

Cet objectif 4.5 sert principalement à trouver un routage qui se rapproche le plus du routage de base défini pour les paquets correspondants aux différents domaines de l'ensemble PR. En effet, pour chaque domaine d'application pr, on essaie de maximiser les affectations à 1 aux variables x_{pr}^{e} qui concernent les arcs e anciennement utilisés dans le routage de base. De plus on affecte une valeur -1 aux $coef_{pr}^{e}$, dont les arcs correspondants n'appartiennent pas au routage de base. Ces affectations à -1 permettront d'obtenir un routage qui minimise le nombre d'arcs à traverser de la source vers la destination d'un domaine pr.

L'ILP, présenté ci-dessus, permet principalement la limitation et la garantie de la bande passante et ne prend pas en compte la mise en oeuvre des points d'acheminement.

Supposant qu'on veut forcer le passage par un commutateur s pour un domaine pr. Nous avons tout d'abord pensé à ajouter une contrainte qui permet de forcer à 1 l'une des variables de décision associées aux arcs issus de s.

Cependant, sur de grands réseaux, cette méthode s'est avérée incorrecte. Elle ne permet pas de forcer le passage par les points d'acheminement. La résolution de l'ILP avec une telle contrainte a produit deux chemins :

1. Le premier chemin va de la source de pr vers sa destination sans passer par s.
2. Le deuxième chemin forme un circuit qui sort de s pour y revenir dès que possible.

Cette anomalie est illustrée dans la Figure 4.2, en appliquant l'approche susmentionnée sur un exemple de réseau contenant 184 commutateurs. Les deux chemins sont indiqués par la couleur rouge.

Pour pallier ce problème, nous avons décomposé la recherche d'un routage pour un domaine de paquets en recherche de sous-chemins qui vont constituer le routage. Par exemple, supposons que nous cherchons un routage pour les paquets du domaine pr_1 suivant tout en forçant le passage par le commutateur s_2 :

$$pr_1 = (src = \ '000',\ dst =' 100')$$

La recherche d'un tel routage revient à trouver les deux chemins suivant :

1. un chemin menant de l'hôte '000' (source de pr_1) vers le commutateur s_2 et
2. un chemin menant de commutateur s_2 vers l'hôte '100' (destination de pr_1).

Ainsi, nous pouvons trouver le chemin désiré en fournissant les deux domaines suivants à notre ILP :

1. $pr_{1-1} = (src = \ '000',\ dst = s_2)$ et
2. $pr_{1-2} = (src = \ s_2,\ dst =' 100')$.

D'une façon générale, supposons la règle suivante qui spécifie les points d'acheminement pour les paquets ayant host1 comme source et host2 comme destination :

(src=host1, dst=host2) : waypoint(s_1, .. ,s_n)

La recherche d'un chemin forçant la séquence des points d'acheminement (s_1, .. ,s_n) se ramène à déterminer les chemins élémentaires pour les domaines suivants :

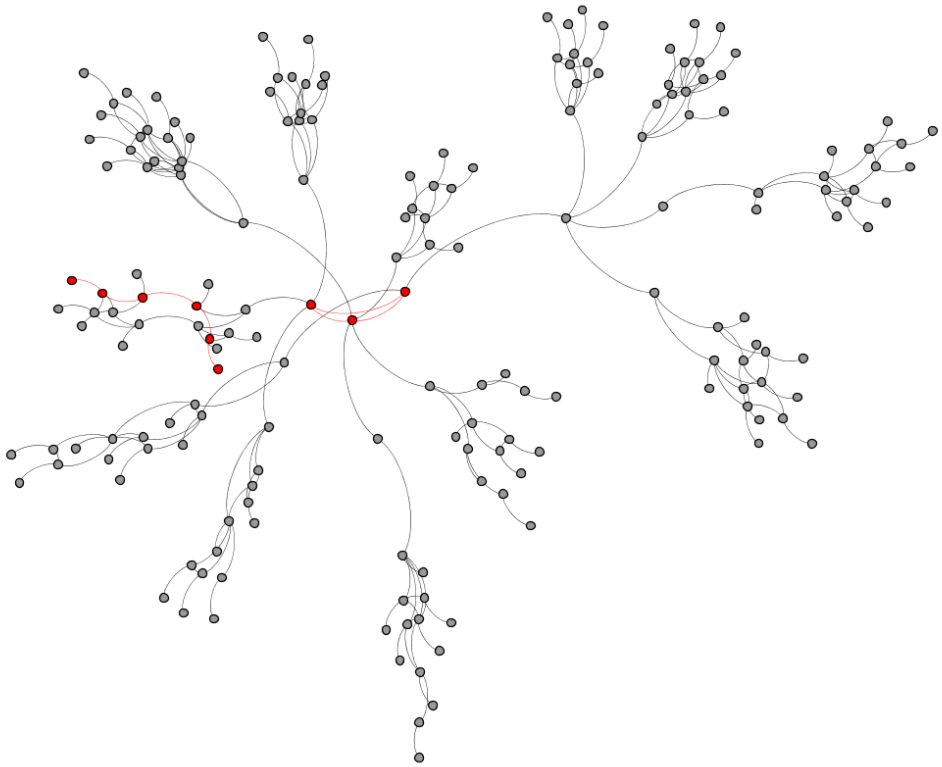


Figure 4.2 Un exemple de réseau illustrant l'anomalie trouvée

{(src=host1, dst=s_1) ; (src=s_1, dst=s_2) ; .. ; (src=s_{n-1} ,dst=s_n)}

Toutefois, cette méthode ne supporte pas le fait de vouloir imposer le choix de forcer un parmi plusieurs points d'acheminement dans un niveau donné. Ce cas s'illustre par la règle suivante à appliquer sur le réseau de la Figure 4.1 :

(src='000', dst='011', srcPort=80) : waypoint($s_2|s_3$, s_4)

Ceci peut poser un problème vu qu'on peut diviser le problème de deux façons. La première division prend en compte le point d'acheminement s_2 tandis que la deuxième considère seulement le point d'acheminement s_3 :

1. {(src='000', dst=s_2) ;(src=s_2, dst=s_4) ;(src=s_4, dst='011')}
2. {(src='000', dst=s_3) ;(src=s_3, dst=s_4) ;(src=s_4, dst='011')}

Pour remédier à ce problème, nous avons étendu notre méthode afin d'assurer la continuité dans chaque niveau de points d'acheminement. La continuité est assurée si la destination d'un chemin élémentaire correspond à la source de chemin élémentaire qui le suit. L'extension est appelée transducteur. Tout en restant dans le même exemple, on dénote le domaine (src='000', dst='011') par pr_2. Considérons le requis suivant qui impose aux paquets correspondants à pr_2 de passer par s_2 ou s_3.

pr_2 : waypoint$(s_2|s_3)$

Comme mentionné ci-dessus, on commence par la division du problème de recherche de routage pour les paquets correspondants à pr_2 en la recherche des deux chemins élémentaires suivants qui correspondent respectivement aux domaines pr_{2-1} et pr_{2-2} :

1. Un chemin allant de '000' (source de pr_2) vers $(s_2|s_3)$; pr_{2-1}=(src='000', dst=$(s_2|s_3)$)
2. Un chemin allant de $(s_2|s_3)$ vers '011'(destination de pr_2) ; pr_{2-2}=(src=$(s_2|s_3)$, dst='011')

Comme l'illustre la Figure 4.3, on ajoute deux nouveaux noeuds, à savoir VD et VS. Le noeud VD consiste en un noeud virtuel qui sert comme destination du chemin correspondant à pr_{2-1}. De son côté, le noeud virtuel VS sert aussi comme source du chemin correspondant à pr_{2-2}. De ce fait, le problème de recherche des deux chemins élémentaires se ramène à chercher les deux chemins élémentaires suivants :

1. Un chemin allant de '000' vers VD ; pr_{2-1}=(src='000', dst=VD)
2. Un chemin allant de VS vers '011' ; pr_{2-2}=(src=VS, dst='011')

Par la suite, on ajoute deux contraintes à notre ILP afin d'assurer la continuité entre les chemins élémentaires. Ces contraintes sont appelées contraintes de dispatching. Elles servent principalement à assurer les égalités suivantes :

1. $x_{pr_{2-1}}^{e_{s_2-VD}} = x_{pr_{2-2}}^{e_{VS-s_2}}$
2. $x_{pr_{2-1}}^{e_{s_3-VD}} = x_{pr_{2-2}}^{e_{VS-s_3}}$

Ces contraintes permettent d'assurer que si le premier chemin élémentaire va vers s_2 alors le deuxième chemin élémentaire commence par s_2 et vice versa. Sinon, si le premier chemin élémentaire va vers s_3 alors le deuxième chemin élémentaire commence par s_3 et vice versa.

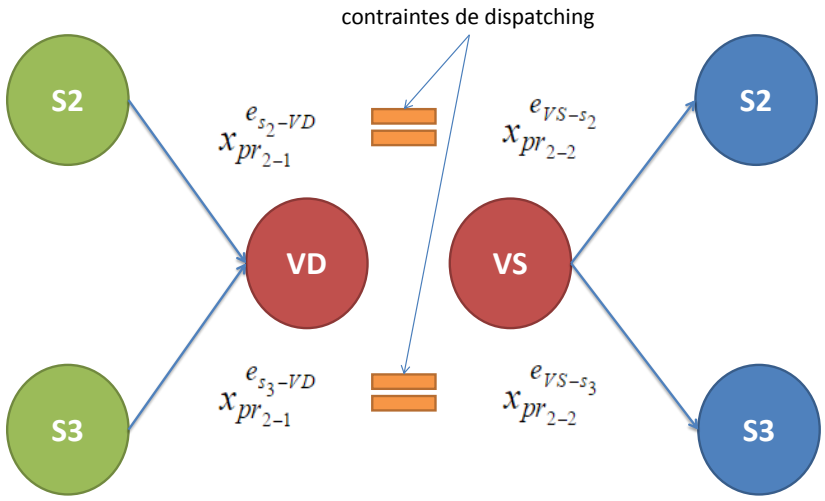


Figure 4.3 Un exemple de transducteur

4.4 Placement des règles

Étant donné un réseau N et une liste de règles R exprimant une politique globale, notre objectif est de trouver un placement des règles dans les commutateurs du réseau tout en respectant les ressources disponibles qui sont en l'occurrence les capacités des commutateurs.

Dans le cas où les capacités des commutateurs ne suffiraient pas pour placer toutes les règles, nous essayerons de placer un maximum de règles.

Chaque règle est constituée d'un domaine d'application et des actions qui s'appliquent lorsqu'un paquet est dans le domaine de cette règle. Les règles peuvent contenir tous les types d'actions qui peuvent être appliquées par un commutateur compatible OpenFlow. Ces types d'actions sont illustrées dans le chapitre des préliminaires.

À titre d'exemple, la Figure 4.4 montre une liste de règles qui concernent le réseau de la Figure 4.1 et ayant des actions de type 'accept' et 'drop' qui mettent en oeuvre une politique de contrôle d'accès. Ces règles sont classées par ordre descendant de priorité et utilisant le caractère générique '*'.

Renforcer la politique illustrée par une règle exprimée moyennant le caractère générique '*' peut ne pas être satisfait en plaçant cette règle dans un seul commutateur. Par exemple, vouloir appliquer la règle suivante sur le réseau de la Figure 4.1 ne peut pas se faire en la

r_1 : src=00*, dst=100: drop
r_2 : src=001, dst=101: accept
r_3 : src=000, dst=110: drop
r_4 : src=00*, dst=***: accept

Figure 4.4 Liste de règles de contrôle d'accès classées par ordre de priorité descendant

plaçant dans un seul commutateur vu qu'il n'y a aucun commutateur qui couvre tous les chemins qui lient les hôtes.

r : src=***, dst=*** : accept

De ce fait, une règle générique peut être divisée en plusieurs règles spécifiques. Chaque règle spécifique aura une source et une destination uniques. De ce fait, cette règle spécifique peut être placée sur n'importe quel commutateur qui se trouve sur le chemin liant sa source à sa destination. Ce chemin est appelé chemin de placement. Il est défini par la politique de routage existante et peut être récupéré par la fonction $routingSwitches$.

Par exemple, la règle r_1 de la figure précédente peut être exprimée moyennant les deux règles spécifiques suivantes, en remplaçant le caractère générique par les valeurs correspondantes.

r_{11} : src=000, dst=100 : drop et

r_{12} : src=001, dst=100 : drop.

Le chemin de placement de la règle spécifique r_{11} est le chemin retourné par la fonction $routingSwitches('000',\ '100') = (s_1,\ s_2,\ s_4)$.

Dans toute la suite, une règle spécifique est appelée portion de règle. r_{11} et r_{12} sont les portions de la règle r_1. Les portions appartenant à la même règle sont appelées portions soeurs. Étant donné une liste de règles, on construit un espace d'entêtes qui illustre la distribution des règles dans cet espace.

L'espace des entêtes est une représentation géométrique des entêtes des paquets où chaque entête est représentée comme un point ayant les coordonnées des valeurs affectées aux champs de cet entête. Chaque champ de l'entête est représenté par une dimension de l'espace.

Les règles de la Figure 4.4 sont exprimées sur deux dimensions, à savoir la source 'src' et la destination 'dst'. Ceci nous ramène à construire un espace des entêtes bidimensionnel. L'espace correspondant aux règles de la Figure 4.4 est illustré dans la Figure 4.5. Chaque dimension contient toutes les valeurs possibles qui résultent du remplacement du caractère générique '*'. Chaque case représente un point de l'espace qui a comme coordonnées, une

source et une destination spécifiques. Une portion de règle correspond à un point dans l'espace des entêtes. La distribution des règles dans l'espace permet d'éliminer les redondances et de déterminer quelle portion de règle correspond à chaque point. Si deux portions de règles correspondent au même point, alors on considère la portion ayant la règle la plus prioritaire. À titre d'exemple, la distribution de la règle r_4 dans l'espace suivant est désignée par la couleur rose. Les portions de cette règle devaient couvrir tout l'espace. Cependant, ils coïncident avec les portions des règles r_1 (vert), r_2 (gris) et r_3 (bleu) respectivement dans les ensembles de points {(000, 100), (001, 100)} ; {(001, 101)} et {(000, 110)}. Finalement, on considère les portions de ces règles vu qu'elles sont plus prioritaires. Le chemin de placement de chaque portion règle est représenté entre parenthèses dans la Figure 4.5. Par exemple, la portion p41 de la règle r_4 peut être placée sur les commutateurs du chemin (s_1, s_3).

En résumé, la représentation des règles dans l'espace des entêtes permet de déterminer le domaine de chaque règle en tenant compte de sa priorité.

destinations

sources \ Hosts	010	011	100	101	110
000	p41(s1,s3)	p42(s1,s2,s4)	p11(s1,s2,s4)	p45(s1,s2)	p31(s1,s2,s5)
001	p43(s1,s3)	p44(s1,s2,s4)	p12(s1,s2,s4)	p21(s1,s2)	p46(s1,s2,s5)

Figure 4.5 Distribution des règles dans l'espace des entêtes bidimensionnel

L'espace des entêtes peut être multidimensionnel dans le cas où les règles sont exprimées sur plusieurs dimensions. Par exemple la règle suivante est exprimée sur 4 dimensions, à savoir l'adresse de la source 'srcIP', l'adresse de la destination 'dstIP', le port source 'srcPort' et le port destination 'dstPort' :

srcIP=100, dstIP=001, srcPort=80, dstPort=80 : drop

Un meilleur placement des règles possède les caractéristiques suivantes :

1. Il maximise le nombre de règles placées dans le cas où les capacités disponibles sur les commutateurs ne permettent pas de placer toutes les règles.
2. Il consomme le minimum de capacité sur les commutateurs dans le cas où on a suffisamment de capacités pour placer toutes les règles.

3. Il rapproche les règles des sources des flux qui leur correspondent. Ceci peut servir à minimiser le nombre de commutateurs à traverser avant qu'un paquet soit traité par une règle qui lui correspond.

Notre problème de placement de règles peut être réduit à un placement des portions de règles tout en respectant les caractéristiques susmentionnées.

La mise en oeuvre des deux premières caractéristiques en se basant sur le concept des portions de règles est faite en satisfaisant les deux objectifs suivants.

1. Maximiser le nombre de portions de règles placées et
2. rassembler un maximum de portions appartenant à une même règle dans le même commutateur, étant donné que ces portions seront exprimées sous forme d'une seule règle. À titre d'exemple, si les portions p11 et p12, de l'exemple précédent, sont placées dans le même commutateur, alors elles seront exprimées sous forme d'une seule règle, à savoir r_1 dans ce cas. On consommera ainsi une seule unité de capacité de ce commutateur. Dans le cas contraire, les deux portions seront placées chacune dans un commutateur et on consommera deux unités de capacité (une dans chaque commutateur). De ce fait, maximiser le placement des portions d'une règle dans le même commutateur permet de consommer moins d'unités de capacité. Ceci permettra de placer plus de règles.

En résumé, le problème de placement des règles est résolu selon une granularité très fine. En effet, nous divisons ces règles en des portions ayant chacune un chemin de placement unique et nous essayons de placer ces portions tout en respectant des caractéristiques qui visent à maximiser le nombre de règles placées.

Les deux sous-sections suivantes visent à résoudre notre problème de placement de deux façons. La première sous-section utilise un programme linéaire multiobjectif en nombres entiers(MOILP). Ce programme sert à placer les règles tout en respectant les deux premières caractéristiques d'un meilleur placement. La deuxième sous-section utilise un algorithme basé sur le problème de 'Minimum Cost Maximum Flow' qui permet de résoudre le placement en peu de temps. Cet algorithme va considérer les trois caractéristiques d'un meilleur placement.

4.4.1 Placement moyennant un programme linéaire multiobjectif en nombres entiers (MOILP)

Formalisation

Le programme MOILP proposé utilise les notations suivantes :

1. *portions* dénote l'ensemble des portions à placer.
2. $\forall\ r\ \in\ R:\ r.portions$ dénote l'ensemble des portions de la règle r.
3. $\forall\ i\ \in\ portions : i.src$ dénote la source du domaine d'application de la portion i.
4. $\forall\ i\ \in\ portions : i.dst$: dénote la destination du domaine d'application de la portion i.

Par exemple, les règles de la Figure 4.4 forment l'ensemble R des règles définissant une politique globale. Les cases du tableau de la Figure 4.5, représentant l'espace des entêtes correspondant aux règles de la Figure 4.4, forment l'ensemble *portions* à placer. Les portions appartenant à la règle r_1 sont $r_1.portions = \{p11,\ p12\}$. $p11.src = 000$ et $p11.dst = 100$.

Notre programme linéaire en nombres entiers utilise les variables de décisions suivantes :

1. $p\ {}_i^s$: est une variable de décision binaire qui indique le placement de la portion i dans le commutateur s. ${p_i}^s = 1$ si la portion de règle i est placée dans le commutateur s. Dans le cas contraire, ${p_i}^s = 0$. Par exemple, la portion $p11$ du tableau de la Figure 4.5 aura les variables suivantes $\{p_{p11}^{s_1},\ p_{p11}^{s_2},\ p_{p11}^{s_4}\}$ qui désignent la possibilité de placer $p11$ dans les commutateurs s_1, s_2 et s_4.

1. $u\ {}_r^s$: est une variable de décision binaire qui indique si des portions de la règle r sont placées dans le commutateur s. ${u_r}^s = 1$ si on a des portions de r qui sont placées dans le commutateur s. Dans le cas contraire, ${u_r}^s = 0$. Par exemple, si la portion $p11$ est placée dans s_1 alors on aura $u_{r_1}^{s_1} = 1$.

Les contraintes de notre MOILP :

$$\forall\ r\ \in\ R:\ \forall\ i\ \in\ r.portions : \forall\ s\ \in\ routingSwitches(i.src,\ i.dst):\ p\ {}_i^s \leq u\ {}_r^s \tag{4.6}$$

$$\forall\ i\ \in\ portions, \sum_{s\ \in\ routingSwitches(i.src,\ i.dst)} p\ {}_i^s \leq 1 \tag{4.7}$$

$$\forall\ s\ \in\ S, \sum_{r \in R} u\ {}_r^s \leq cap(s) \tag{4.8}$$

La contrainte 4.6 considère qu'une règle r est placée dans un commutateur s si une ou plusieurs portions i de cette règle sont placées dans le même commutateur. Cette notion sert plus tard dans le compte des règles placées dans un commutateur vu que nous comptons seulement le nombre de règles et non des portions. En effet, les portions appartenant à une même règle et placées dans le même commutateur seront exprimées moyennant une seule règle et consommeront ainsi une seule unité de capacité du commutateur.

La contrainte 4.7 indique qu'une portion de règle ne doit pas être installée dans plus qu'un commutateur. La contrainte 4.8 indique que le nombre de règles placées dans chaque commutateur ne doit pas excéder sa capacité.

Notre problème de placement optimise les deux fonctions suivantes :

$$f_1(u) = \sum_{r \in R} \sum_{s \in S} u_r^s \tag{4.9}$$

$$f_2(p) = \sum_{i \in portions} \sum_{s \in S} {p_i}^s \tag{4.10}$$

La fonction 4.9 est la somme des variables exprimant le placement d'une règle dans un commutateur donné. La valeur de cette fonction exprime la totalité de la capacité consommée dans les commutateurs. La fonction 4.10 consiste en la somme de la totalité des portions à placer.

Le but de notre programme linéaire en nombres entiers est de placer le maximum de portions de règles tout en minimisant l'espace consommé. Ceci se traduit par la minimisation de f_1 (4.9) et la maximisation de f_2 (4.10). Le placement se ramène, donc, à l'optimisation multiobjective suivante :

$$min \{ f_1(u), -f_2(p)\} \tag{4.11}$$

sous les contraintes

$$(4.6), (4.7), (4.8)$$

Combinaison des objectifs

Tout au long de cette section, nous présentons les concepts liés à l'optimisation multiobjective puis nous expliquons l'approche adoptée pour mettre en oeuvre les objectifs de notre problème.

Un programme linéaire en nombres entiers, contenant m fonctions objectif, est défini de la façon suivante :

$$min\ F(x) = \{f_i(x) = {ct_i}^T\ x\ ,\ i = 1, 2, ..., m\} \tag{4.12}$$

sous les p contraintes :

$$\sum_{j=1}^{n} a_{k\ j} x_j \leq b_k \qquad\qquad k = 1, 2, ..., p$$

$x = \{x_j\}_{1\ \leq\ j\ \leq\ n}$ dénote le vecteur des variables de décision. Chaque x_j est un entier.

$ct_i = \{ct_i\}_{1\ \leq\ j\ \leq\ n}$ dénote le vecteur des coefficients relatifs à la fonction objectif f_i.

Considérons les définitions suivantes :

1. Soit X l'ensemble des solutions réalisables du programme linéaire 4.12, $X = \{x \in\ Z^n : A\ x \leq b\}$; $A = \{a_{i\ j}\}_{1\leq\ i\ \leq\ p\ et\ 1\leq\ j\ \leq\ n}$et $b = \{b_j\}_{1\ \leq\ j\ \leq\ p}$
2. Soit Y l'ensemble des points réalisables dans l'espace des objectifs, $Y = \{F(x),\ x \in X\}$.

Considérons x' et x'', deux solutions réalisables de l'ensemble X. On dit que x' **domine faiblement** $x'' \Leftrightarrow$ la condition 4.13 est satisfaite :

$$\forall\ i\ \in \{1, 2, ..., m\}\ : f_i\ (x') \leq f_i\ (x'') \tag{4.13}$$

Dans d'autres mesures, on dit que x' **domine** $x'' \Leftrightarrow$ les conditions 4.13 et 4.14 soient satisfaites :

$$\exists\ i \in \{1, 2, ..., m\} : f_i\ (x') < f_i\ (x'') \tag{4.14}$$

Une solution $x^* \in X$ est dite **faiblement efficace** si et seulement s'il n'y a aucun $x \in X$ tel que $f_i(x) < f_i(x^*),\ \forall\ i\ \in \{1, 2, ..., m\}$.

Une solution $x^* \in X$ est dite **efficace** s'il n'y a aucun autre vecteur $x \in X$ tel que x domine x^*. $F(x^*)$ dénote un point non dominé de l'espace des objectifs. L'ensemble des points non dominés est appelé front de Pareto. La Figure 4.6 représente l'espace des objectifs pour un problème d'optimisation contenant deux fonctions objectif f_1 et f_2.

Selon la revue de (Ehrgott and Gandibleux, 2000), les méthodes de résolution des problèmes multiobjectif peuvent être classées en deux grandes catégories. La première catégorie concerne les méthodes exactes. La deuxième catégorie concerne les méthodes approximatives. Les mé-

thodes approximatives se basent sur des heuristiques et des métaheuristiques afin de générer des solutions approximatives en un temps raisonnable. Dans cette section, nous nous intéressons seulement aux méthodes exactes. Un algorithme basé sur heuristiques sera présenté dans la prochaine section. Les tests vont montrer l'efficacité de cet algorithme en terme de temps de calcul.

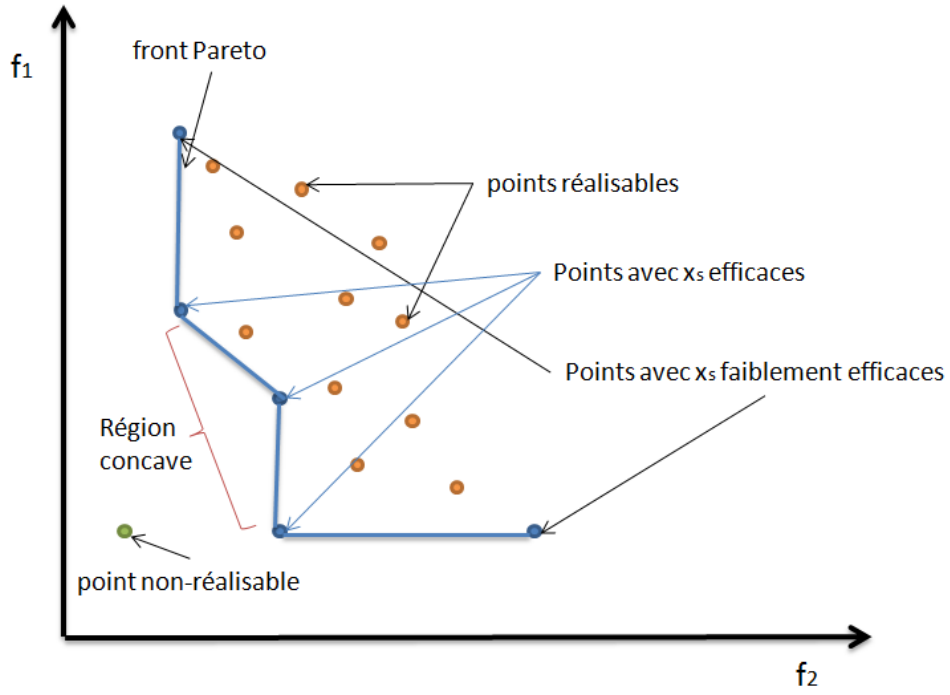


Figure 4.6 Exemple d'espace des objectifs bidimensionnel

Les méthodes scalaires constituent la partie majeure des méthodes exactes. Le travail de (Ehrgott, 2006) présente les différentes méthodes scalaires applicables aux problèmes d'optimisations multiobjectif en nombres entiers (MOILP). Ce type de méthodes vise principalement à résoudre le problème d'optimisation en utilisant une seule fonction objectif. Les deux méthodes scalaires les plus populaires sont

1. La méthode de la somme pondérée (weighted sum method) et
2. la méthode ε-contrainte (ε-constraint method).

La méthode de la somme pondérée consiste à combiner les fonctions objectif dans une seule fonction objectif en affectant un poids σ_ià chaque fonction. L'optimisation se ramène ainsi à l'expression suivante :

$$min \sum_{i=1}^{m} \sigma_i \ f_i(x) \tag{4.15}$$

La minimisation 4.15 produira des solutions efficaces si $\sigma_i > 0$; $i = 1, 2, ..., m$. Cependant, les solutions efficaces, se trouvant à l'intérieur de la coque convexe (dans la partie concave), ne peuvent pas être trouvées par la méthode de la somme pondérée. De ce fait, les solutions efficaces sont classées de la façon suivante :

1. Solutions efficaces supportées : solutions efficaces produites par la méthode de la somme pondérée si les conditions suivantes sont respectées : $0 < \sigma_i < 1$; $i = 1, 2, ..., m$ et $\sum_{i=1}^{m} \sigma_i = 1$.
2. Solutions efficaces non supportées : solutions efficaces qu'on ne peut pas trouver par la méthode de la somme pondérée.

La méthode ε-contrainte permet de trouver toutes les solutions efficaces ou faiblement efficaces (supportées et non supportées). Cette méthode minimise seulement un objectif parmi les m objectifs et met les $m - 1$ autres en tant que contraintes. Le problème d'optimisation se transforme comme suit :

$$min \;\; f_j\,(x)$$

sous les contraintes :

$$x \;\in\; X$$

$$f_i \leq \varepsilon_i \;; i \in [1...m] \backslash \{j\}$$

Explorer les solutions du front Pareto revient à varier les ε_i. Ces dernières années ont vu l'apparition de plusieurs implantations de la méthode ε-contrainte. L'algorithme AUGMECON présenté dans l'oeuvre de (Mavrotas, 2009), consiste en un exemple d'implantation qui vise à trouver un maximum de solution du front Pareto. Afin d'illustrer le fonctionnement de la méthode ε-contrainte, nous présentons l'implantation de la Figure 4.7 visant à minimiser deux fonctions objectif, à savoir $f_1$et f_2. Cette implantation commence par trouver une solution optimale à la fonction objectif f_1. Ensuite, elle essaie de restreindre l'espace des solutions en modifiant, à chaque itération, la valeur de ε_2.

La résolution de notre problème est faite moyennant la méthode ε-contrainte. De ce fait, l'optimisation multiobjectif 4.11 est exprimée par le programme monoobjectif suivant :

$$min \;\; f_1\,(u) \tag{4.16}$$

sous les contraintes :

$$u \;\in\; U,\; p \;\in P$$

$$f_2(p) \leq \varepsilon_2$$

où U et P dénotent respectivement l'espace des solutions des variables u et p.

Ce programme peut se résoudre par l'algorithme précédent afin de produire un ensemble

```
epsilonContrainteClassique(){
        Déterminer une solution optimale x^1 de f_1 ;
        solutions = x^1 ;
        ε_2 = f_2(x^1) – δ ;
        tant que x* = min_{x ∈ X} {f_1(x) | f_2(x) ≤ ε_2} est faisable{
                solutions ← solutions ∪ x* ;
                ε_2 ← f_2(x^1) – δ ;
        }
}
```

Figure 4.7 Implémentation de la méthode ε-contrainte

de solutions efficaces ou faiblement efficaces. Cet ensemble peut être utilisé ultérieurement afin de faire un choix manuel du placement qu'on juge raisonnable. Cependant, notre outil devrait résoudre le placement automatiquement, sans la participation d'un tiers.

De ce fait, nous avons décidé de favoriser les solutions qui maximisent le nombre de portions de règles placées. Pour ce faire, nous avons défini une implantation, de la méthode ε-contrainte, qui permet de trouver une seule solution efficace (ou faiblement efficace) qui maximise le nombre de portions placées.

En effet, notre implantation commence par chercher un placement à toutes les portions tout en minimisant l'espace consommé. Si un tel placement n'est pas faisable, alors elle procède à une recherche dichotomique qui vise à maximiser les portions de règles placées. La Figure 4.8 illustre notre implantation. La fonction **"résoudreILPAvec(ε)"** permet de résoudre le programme monoobjectif précédent avec la valeur ε_2 donnée comme paramètre. La fonction **"rechercheDichotomique(lb, ub)"** cherche, d'une façon dichotomique, une solution qui se rapproche le plus de la solution maximisant le nombre de portions de règles placées. Cette fonction prend comme paramètres :

1. lb : dénote la limite inférieure de la région de recherche.
2. ub : dénote la limite supérieure de la région de recherche

La variable δ définit la précision. En effet, une solution favorise la maximisation des portions de règles placées, si elle se trouve dans la région de recherche $[lb, ub]$ tel que $\frac{ub-lb}{2} \leq \delta$.

```
calculDeSolutionFavorisantUnMaximumDePortionsPlacées(){
        ε2 = – nombreDePortions ;
        x = résoudreILPAvec( ε2 );
        si(x!=Nil){
                retourner x ;
        } sinon {
                retourner rechercheDichotomique( ε2 , 0);
        }
}
```

(a)

```
rechercheDichotomique( lb , ub ){
        ε2 = (lb + ub) ÷ 2 ;
        x = résoudreILPAvec( ε2 );
        si ( x != Nil && (ub – lb) ÷ 2 ≤ δ ){
                retourner x ;
        } sinon si ( x != Nil && (ub – lb) ÷ 2 > δ ){
                retourner rechercheDichotomique( lb , ε2 );
        } sinon si( x == Nil && (ub – lb) ÷ 2 > δ ){
                retourner rechercheDichotomique( ε2 , ub )
        } sinon {
                retourner Nil;
        }
}
```

(b)

```
résoudreILPAvec( ε2 ){
        si( x = min x ∈ X {f1(x) | f2(x) ≤ ε2 } est faisable){
                retourner x ;
        } sinon {
                retourner Nil;
        }
}
```

(c)

Figure 4.8 Implantation de la méthode $\varepsilon - contrainte$ favorisant la maximisation de nombre de portions de règles placées

La Figure 4.9 montre un exemple d'exécution de notre implantation de la méthode $\varepsilon -$ *contrainte* sur notre programme multiobjectif. Cette exécution concerne le cas où la méthode ne parvient pas à placer la totalité des portions de règles données en entrée.

La recherche d'une solution exacte moyennant notre MOILP se réduit à la résolution de plusieurs ILP qui est NP-Difficile. De ce fait, la recherche de la solution peut prendre beaucoup de temps alors que nous cherchons une méthode rapide afin de nous adapter rapidement aux changements des politiques de gestion exprimées moyennant des règles.

La sous-section suivante fournit un algorithme basé sur des heuristiques qui permet de placer les règles. Les tests vont montrer l'efficacité de cet algorithme en terme de temps d'exécution.

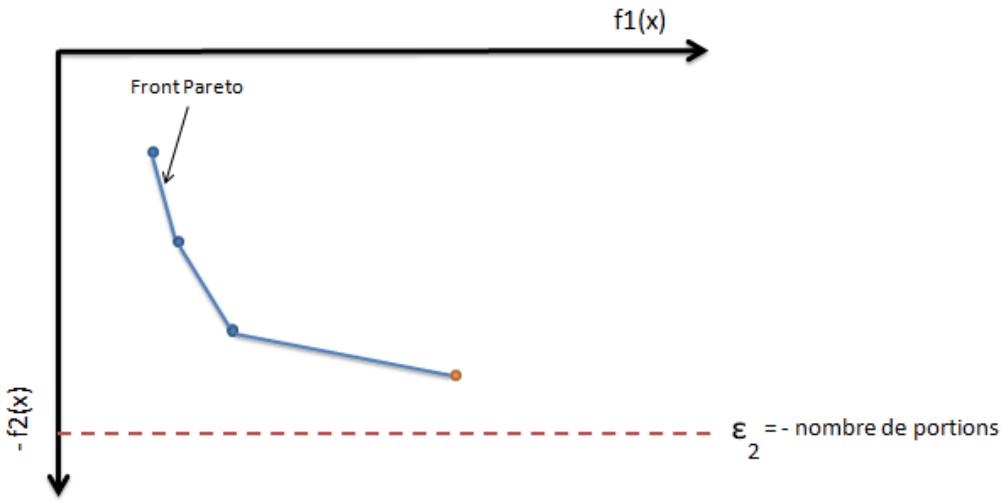
4.4.2 Placement moyennant un algorithme basé sur des heuristiques

L'algorithme de placement se base sur l'approche définie dans le travail de (Gabay et al., 2014). Ce travail propose une approche générique pour résoudre le problème de placement des éléments (packing problem). Cette approche peut s'appliquer à plusieurs problèmes tels que le 'bin packing'. En effet, le problème de placement se base sur un graphe biparti appelé graphe de compatibilité. Ce graphe, dénoté par G, est composé d'un ensemble de noeuds V et un ensemble d'arcs A. L'ensemble V est défini comme l'union de deux ensembles I et B. I dénote le sous-ensemble de noeuds représentants les éléments (items) à placer. B dénote le sous-ensemble de noeuds représentants les bacs (bins) où les éléments seront placés. Un arc $a \in A$ lie un élément $i \in I$ à un bac $b \in B$ si l'élément pouvait être placé dans ce bac. On dit que i est compatible avec b. Cette définition du graphe de compatibilité est utilisée afin de déterminer un placement un à un qui associe un seul élément à un seul bac. La Figure 4.10 illustre un exemple de graphe de compatibilité entre les éléments (i1, i2, i3, i4) et les bacs (b1, b2, b3). L'élément i1 est compatible avec les bacs b1 et b2.

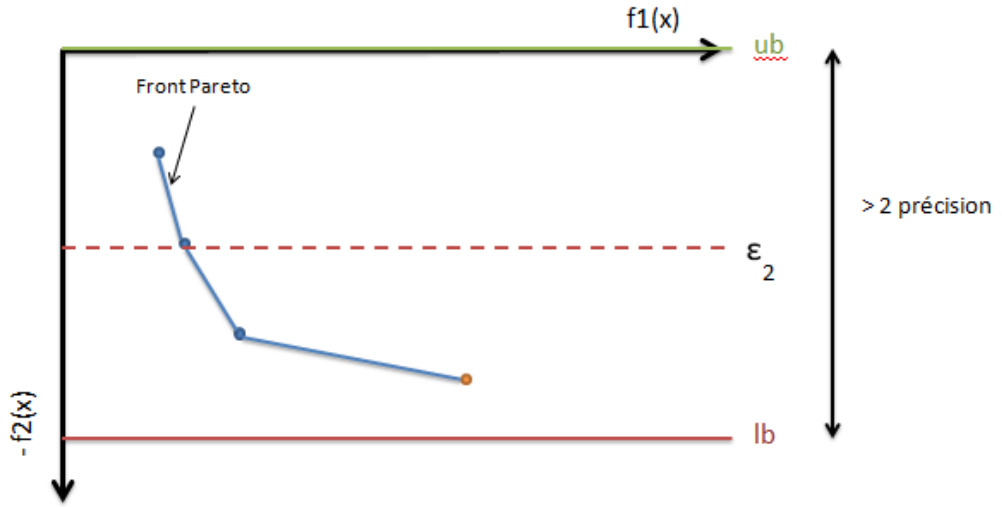
Ce placement peut se réduire au problème de couplage maximum (maximum matching) expliqué dans le chapitre 2.

Le problème de couplage maximum pourrait se réduire en un problème de flot maximum en construisant un nouveau graphe G_f qui enrichit le graphe G par deux noeuds, à savoir o et d.

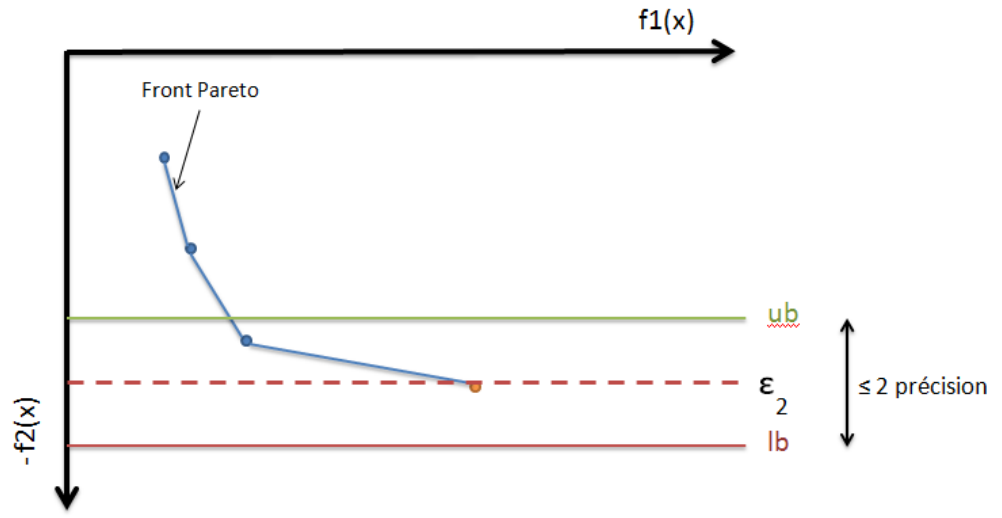
Le noeud o est lié à chaque noeud $i \in I$ tandis que le noeud d est lié à chaque noeud $b \in B$. Ainsi, l'ensemble des arcs A serait égal à l'union de trois sous-ensembles. $A = A_o^I \cup A_I^B \cup A_B^d$. A_o^I dénote le sous-ensemble d'arcs qui lient le noeud o à chaque noeud $i \in I$. A_I^B dénote le



(a) Non-satisfaction de toutes les portions de règles



(b) Solution hors du champ de précision



(c) Solution dans le champ de précision

Figure 4.9 Non-satisfaction de toutes les portions de règles

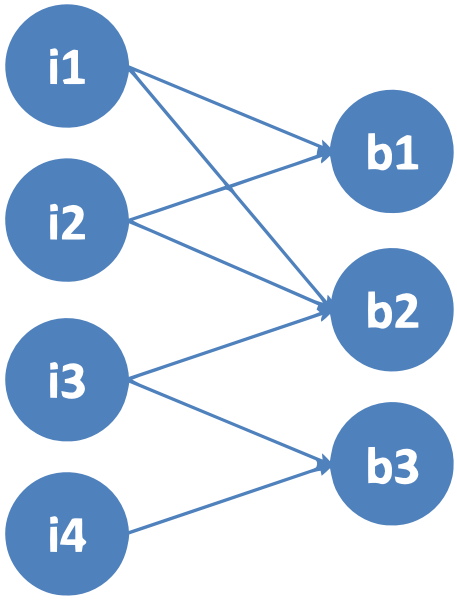


Figure 4.10 Un exemple de graphe de compatibilité simple

sous-ensemble d'arcs qui lient les éléments aux bacs. Finalement, A_B^d dénote le sous-ensemble d'arcs liant les noeuds, représentants les bacs, au noeud d.

La Figure 4.11 montre un exemple de graphe G_f qui correspond au graphe de compatibilité de la Figure 4.10. De ce fait, la recherche d'un couplage maximum revient à trouver un flot maximum entre o et d. Dans le cas du placement un à un, on associe une capacité de 1 à chaque arc a appartenant à l'union des arcs liants o aux éléments et des arcs liants les bacs à d, à savoir $A_o^I \cup A_B^d$. On associe des capacités 8 aux arcs de l'ensemble A_I^B.

Ces capacités assignées aux arcs déterminent le flot maximum qui peut circuler à travers ces arcs. De ce fait, on aura un seul élément affecté à chaque bac. Les affectations correspondent aux flots positifs dans les arcs de sous-ensemble A_I^B. La valeur de flot maximum correspond au couplage maximum et au nombre d'éléments placés. Les auteurs de l'article ont généralisé l'approche afin de supporter des problèmes d'affectations autres que le un à un.

Le graphe G_f peut être modifié afin de supporter le fait qu'un bac puisse supporter plusieurs éléments. À cet effet, la capacité d'un arc $(b,\ d)$ de sous-ensemble A_B^d sera égale à la capacité du bac représenté par le noeud b. Le problème de flot maximum pourrait, désormais, faire circuler sur les arcs A_B^d autant d'unités de flot que la capacité de chaque bac.

Cette généralisation du problème d'affectation permet de l'adapter et de l'utiliser dans notre contexte. En effet, chaque commutateur (représenté par un bac) peut accepter plusieurs

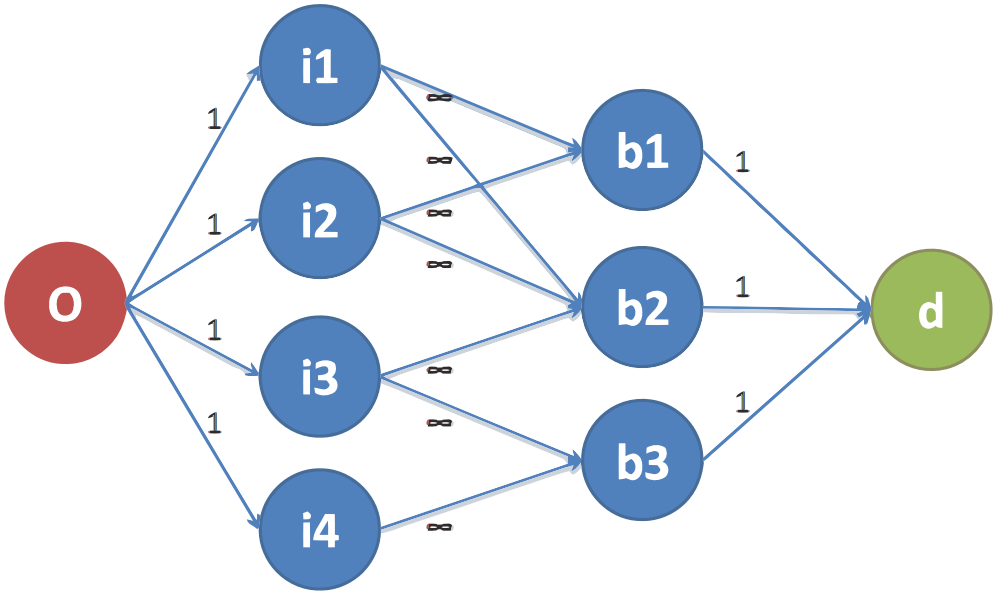


Figure 4.11 Un exemple de graphe de compatibilité par flot

règles (représentées par des éléments). Nous avons donc adopté cette approche de résolution d'affectation basée sur la recherche de flot maximum afin de résoudre notre problème de placement.

Le problème de flot maximum peut être résolu par la méthode de Ford-Fulkerson élaborée dans le travail de (Ford and Fulkerson, 1956). Cette méthode est décrite par l'algorithme de la Figure 4.12.

```
Ford-Fulkerson(){
        tant que(existe_un_chemin_d'augmentation){
                calculer le chemin d'augmentation P ;
                bCap = la capacité goulot d'étranglement de P ;
                augmenter le flot dans P par bCap;
        }
}
```

Figure 4.12 Ford-Fulkerson

Le chemin d'augmentation P consiste en un chemin simple entre o et d dans le graphe résiduel correspondant à G_f. En effet, ce chemin d'augmentation est représenté par la séquence $(a_1, a_{2,}...a_k)$. Aucun arc a_i ne doit être saturé (le flot qui y passe ne dépasse pas sa capacité). Le graphe résiduel est une représentation de G_f marquée par les quantités de flots qui

restent. La capacité goulot d'étranglement d'un chemin P représente le flot maximum qu'on peut envoyer à travers ce chemin. La valeur de cette capacité correspond à la valeur minimum des capacités restantes sur les arcs de ce chemin.

L'augmentation de flot à travers P consiste à envoyer un flot correspondant à la capacité goulet d'étranglement et à mettre à jour le graphe résiduel.

Le problème de flot maximum peut satisfaire notre premier objectif vu qu'il permet de maximiser le nombre de portions de règles qui seront placées. Pour faire face aux deux autres objectifs, nous avons enrichi cette approche en affectant des coûts aux différents arcs de graphe de compatibilité adapté au flot maximum (G_f).

Ainsi, chaque arc $a \in A$ aura, désormais, deux valeurs associées, à savoir u_a et c_a. u_a dénote la capacité de l'arc a. Cette capacité indique la quantité maximale de flot qui peut passer à travers a. c_a dénote le coût associé à l'envoi d'une unité de flot à travers l'arc a.

Ces coûts servent à déterminer les meilleurs chemins d'augmentation à choisir afin de passer le flot. Ce problème est connu sous le nom 'Min Cost Max Flow'. Il a pour but de chercher un flot maximum ayant un coût minimum entre une source o et une destination d dans un graphe donné.

Par conséquent, notre problème de placement des règles se réduit à un problème de 'Min Cost Max Flow' sur le graphe G_f et entre les noeuds o et d.

Le problème de 'Min Cost Max Flow' peut être résolu par l'algorithme 'Cheapest Augmenting Path' (CAP). Cet algorithme peut être implémenté par la méthode de Ford-Fulkerson. Pour ce faire, il suffit de choisir, à chaque étape, le chemin d'augmentation le moins coûteux. Ce chemin d'augmentation peut être considéré comme le plus court chemin minimisant la totalité des coûts.

Avant d'illustrer l'algorithme en détail, on commence par la définition des coûts que nous avons introduits :

$\forall\ a\ \in A_I^B,\ c_a = lp * {f_c}^s + {r_i}^s$. La valeur affectée à c_a, pour les arcs situés entre les noeuds représentants les portions de règles et les noeuds représentants les commutateurs, est définie en fonction ${f_c}^s$ et ${r_i}^s$. ${r_i}^s$ dénote le rang du commutateur de placement potentiel s dans le chemin de placement de la portion de règle i. ${f_c}^s$ représente la fréquence d'apparition de commutateur s dans les différents chemins de placement des portions de la règle c. La maximisation de c_a sert à favoriser le placement des portions de règles appartenant aux mêmes règles. Ceci minimise la capacité à consommer au sein des commutateurs. En effet, les portions appartenant à la même règle et placées dans un même commutateur seront exprimées sous forme d'une seule règle dont le placement ne consomme qu'une place dans le commutateur.

Dans d'autres mesures, le coût c_a considère le placement des règles à proximité des sources concernées par ces règles par le moyen du coût ${r_i}^s$. En conséquence, si deux commutateurs ont la même fréquence ${f_c}^s$, le coût ${r_i}^s$ déterminera le commutateur le plus proche de la source concernée par la portion i. Le fait de multiplier ${f_c}^s$ par lp sert à donner la priorité au critère représenté par ${f_c}^s$. De ce fait, lp est égale à la longueur de plus long chemin de placement $max\ \{r_i^s; i \in portions, s \in S\}$.

Étant donné les définitions des coûts, le meilleur chemin d'augmentation serait un chemin qui maximise c_a pour tous les arcs $a\ \in A_I^B$.

La Figure 4.14 montre le graphe G_f correspondant à notre exemple. Chaque arc a est annoté par le couple ($u_a,\ c_a$) dénotant respectivement la capacité et le coût. Le couple (∞, 3*6+1) illustre le coût 3*6+1. La valeur 3 représente lp, la valeur 6 représente ${f_4}^{s_1}$ et la valeur 1 représente le rang de s_1 dans le chemin de placement de la portion 41.

Toutefois, nous voulons déterminer un chemin d'augmentation qui minimise le coût. De ce fait, la deuxième condition se ramène à une minimisation de $-c_a$ pour tous les arcs $a\ \in A_I^B$.

La détermination de plus court chemin d'augmentation peut s'appuyer sur l'un des algorithmes classiques tels que Dijkstra ou Bellman-Ford. L'algorithme de Bellman-Ford est plus coûteux que celui de Dijkstra, mais permet des coûts négatifs. Afin d'utiliser l'algorithme de Dijkstra, nous transformons les coûts associés aux arcs $a \in A_I^B$ en des coûts positifs de la façon suivante :

$$c_a = max(c_a : a \in A_I^B) - c_a$$

Cette transformation permet d'entamer le problème avec des coûts positifs. Toutefois, des coûts négatifs peuvent être introduits dans le graphe résiduel (expliqué dans le chapitre 2), en essayant d'augmenter le flot. De ce fait, on peut introduire le concept de réduction des coûts. Ce concept permet de modifier les coûts réels afin de les rendre positifs. Selon (Ravindra et al., 1993), le calcul des plus courts chemins en utilisant les coûts réduits donne les mêmes résultats que le calcul basé sur les coûts réels. Pour tout arc $a\ \in A$, le coût réduit ${c_a}^\pi$ est défini comme suit :

$$c_a^\pi = \pi(source(a)) + c_a - \pi(sink(a)) \tag{4.17}$$

Pour chaque noeud $v \in V$, la fonction $\pi(v)$ doit maintenir les coûts positifs ${c_a}^\pi\ t.q\ \ a\ \in A$. L'algorithme de Ford-Fulkerson avec l'heuristique de plus court chemin d'augmentation est illustré dans la Figure 4.13. Pour chaque $v \in V$, $\pi(v)$ est initialisé à 0. Ceci permettra de garder des coûts positifs vu qu'on entame notre problème avec des coûts positifs c_a.

```
procédure minCostMaxFlow{
        Pour chaque v ∈ V : π(v) = 0 ;
        tant que(existe_un_chemin_d'augmentation){
                d = le vecteur des distances des plus courts chemins qui
                partent de o vers tous les autres noeuds selon les coûts
                réduits c_a^π ;
                P = le chemin d'augmentation le plus court;
                bCap = la capacité goulot d'étranglement de P ;
                augmenter_le_flot(P, bCap);
                Pour chaque v ∈ V : π(v) = π(v) + d(v) ;
        }
}
```

Figure 4.13 Algorithme Ford-Fulkerson avec l'heuristique de plus court chemin d'augmentation

Rappelons que l'insertion de plusieurs portions d'une même règle dans le même commutateur ne consomme qu'une seule unité de capacité de ce commutateur. De ce fait, l'augmentation de flot au niveau d'un arc $a \in A_B^d$, ayant une capacité correspondante à celle du commutateur représenté par la source de l'arc a, ne doit se faire que pour les portions n'ayant pas d'autres portions soeurs qui envoient un flot vers cette source.

4.5 Conclusion

Tout au long de ce chapitre, nous avons présenté les approches que nous avons définies afin de mettre en oeuvre les aspects de gestion considérés. Les aspects de la bande passante et de la composition des points d'acheminement sont mis en oeuvre par un même programme linéaire en nombres entiers. Nous avons aussi défini deux méthodes pour placer les règles. L a première méthode se base sur un programme linéaire multiobjectif en nombres entiers. La deuxième méthode se base sur le calcul de flot maximum avec un coût minimum. Le chapitre suivant contiendra un exemple d'exécution de l'algorithme qui matérialise la deuxième méthode. Nous y présentons aussi l'implantation et les tests effectués pour valider nos approches.

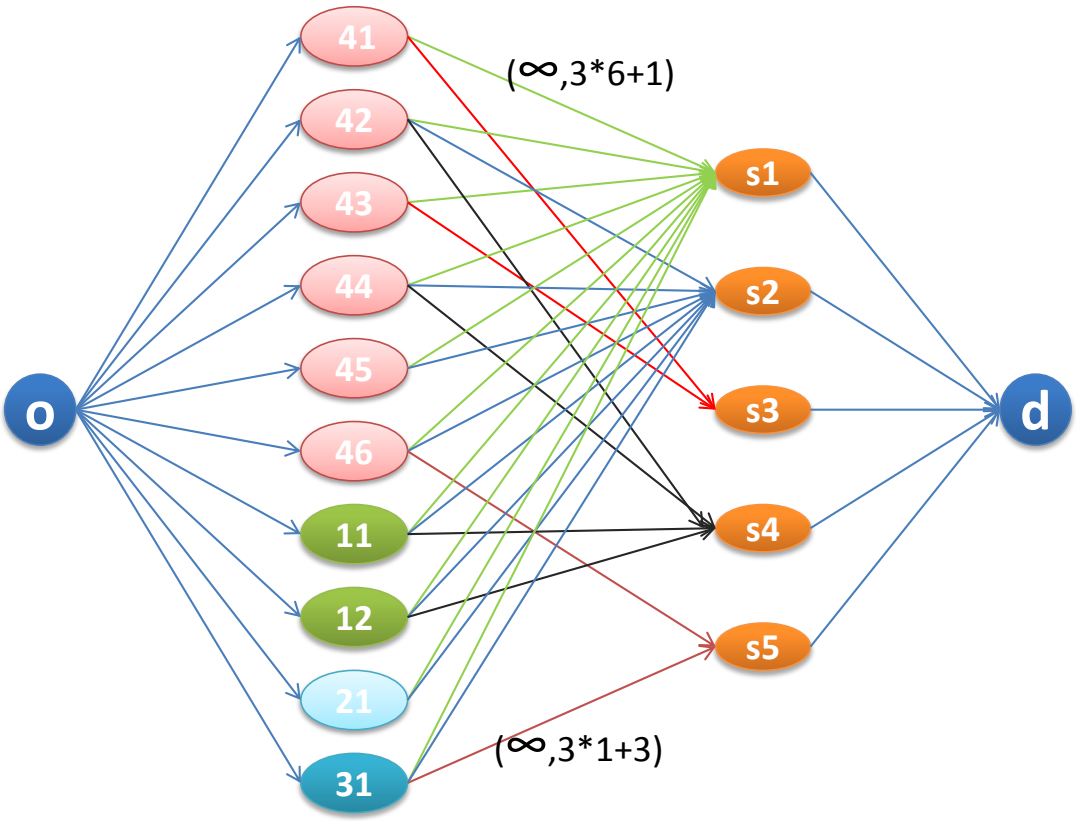


Figure 4.14 graphe G_f correspondant à l'espace des entêtes de notre exemple

CHAPITRE 5 IMPLANTATION ET RÉSULTATS

Dans le chapitre précédent, nous avons présenté les approches que nous avons définies afin de mettre en oeuvre les aspects de gestion considérés. Tout au long de ce chapitre, nous présentons l'outil que nous avons développé pour tester et évaluer nos approches. La première partie de ce chapitre sert à présenter l'implantation de ces approches et à la proposition d'un autre algorithme de placement qui améliorera fortement le temps d'exécution par rapport à l'algorithme, basé sur la recherche de flot maximum avec un coût minimum, tout en gardant presque les mêmes performances.

Dans la deuxième partie, nous présentons l'approche de test de notre outil ainsi que ces performances.

5.1 Implantation

5.1.1 Méthodologie et technique de développement

Le développement de notre outil a été réalisé en se basant sur le modèle des processus de développement agile. Ces processus de développement sont caractérisés par deux aspects, à savoir incrémental et itératif. De ce fait, notre développement s'est effectué sur plusieurs itérations. Durant chaque itération, on ajoute un incrément qui consiste en une fonctionnalité de notre outil. Ainsi, nous avons procédé aux étapes suivantes pour chaque itération :

1. analyse des requis,
2. modélisation,
3. implantation et
4. tests.

Le développement agile permet de se concentrer sur chaque fonctionnalité afin de bien la valider.

Afin de réaliser les étapes d'implantation et de tests de chaque itération, nous avons suivi la technique de développement piloté par les tests (TDD). Cette technique (Beck, 2003) consiste à commencer par l'écriture des tests unitaires avant le développement du code de production mettant en oeuvre la logique métier de l'application.

En effet, chaque fonctionnalité est divisée en plusieurs sous fonctionnalités élémentaires et faciles à tester. Le développement de chaque fonctionnalité élémentaire, selon TDD, consiste à suivre les étapes suivantes :

1. Écrire un test unitaire
2. Exécuter ce test qui doit normalement échouer vu que le code de production n'est pas encore écrit.
3. Écrire le code de production concerné par le test unitaire de la première étape.
4. Exécuter le test qui doit normalement réussir vu qu'on a écrit le code soumis au test dans l'étape précédente.
5. Procéder au remaniement du code de production (refactoring) afin d'améliorer sa qualité.

Le TDD permet d'améliorer la qualité du code en écrivant à chaque étape une seule unité ayant une forte cohésion et facile à tester. Il permet aussi la validation des sous-fonctionnalités ainsi que leur intégration d'une façon itérative.

La Figure 5.1 suivante illustre les itérations de développement de notre outil.

L'itération 0 consiste en l'étape préliminaire durant laquelle nous avons défini l'architecture de notre outil.

Durant l'itération 1, nous avons modélisé et implanté le modèle de données partagé de notre application.

L'itération 2 a servi à la mise en oeuvre des générateurs de bancs de test (topologie et règles).

Durant l'itération 3, nous avons implanté le composant utilitaire qui sert à transformer notre modèle de programme linéaire en un modèle propre à l'outil d'optimisation à utiliser.

L'itération 4 a servi à l'implantation du module de routage qui met en oeuvre les deux premiers aspects de gestion, à savoir la bande passante et les points d'acheminement.

Durant l'itération 5, nous avons implanté le module de placement de règles.

5.1.2 Conception

Architecture

La Figure 5.2 illustre l'architecture de notre outil au moyen d'un diagramme de paquetage.

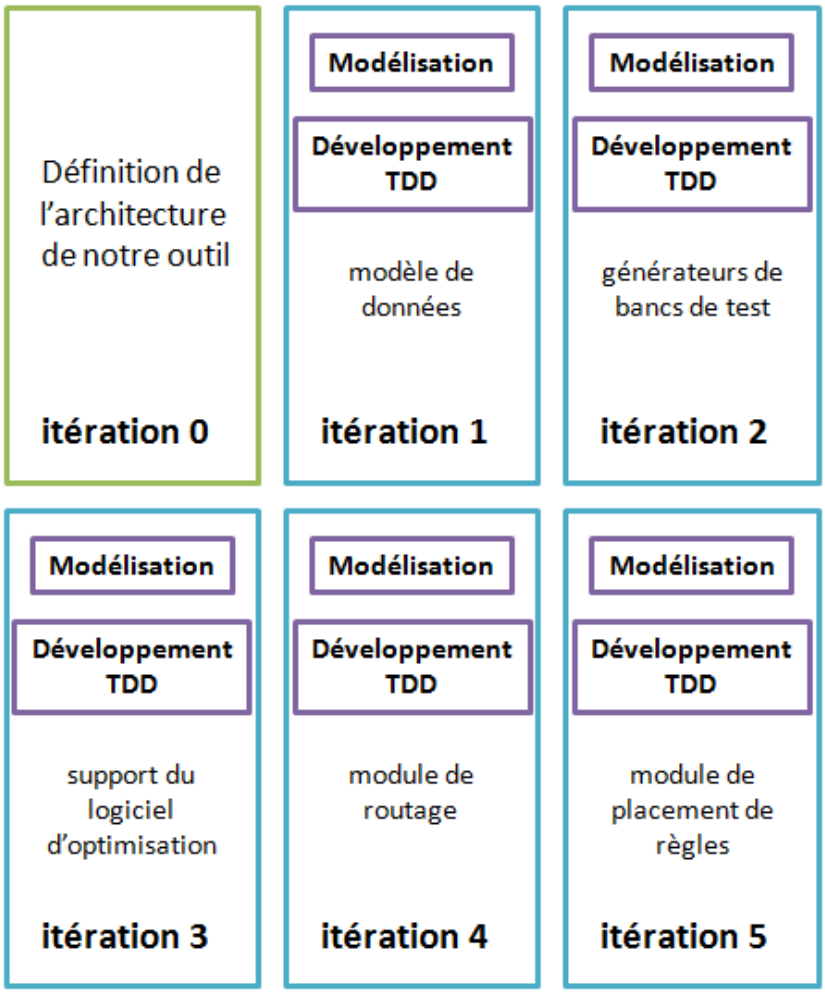


Figure 5.1 itérations de développement de notre outil

En effet, notre outil consiste en trois composants qui définissent sa logique métier ainsi qu'un module utilitaire appelé 'sdnaspects.util'.

Les trois composants principaux sont :

'sdnaspect.model' : Ce composant consiste en le modèle de données de notre outil. Ce modèle représente les entités principales, manipulées dans l'outil, à savoir le réseau et les programmes linéaires. De ce fait, ce composant est constitué de deux paquetages suivants :

1. 'sdnaspect.model.lp' : contient les entités qui représentent un programme linéaire.
2. 'sdnaspect.model.network' : contient les entités qui représentent un réseau.

'sdnaspect.routing' : Ce composant sert à l'implantation des approches de renforcement des aspects de gestion qui sont mis en oeuvre à travers le routage (les points d'acheminement et

la garantie et la limitation de la bande passante). Ce composant se base sur le composant ‘sdnaspects. model’ pour calculer le routage.

‘sdnaspect.placement’ : Ce composant sert à mettre en oeuvre les deux méthodes de placement de règles, à savoir le programme linéaire multiobjectif en nombres entiers et l’algorithme approximatif. De ce fait, ce composant utilise le composant ‘sdnaspects.model’. De plus, ce composant contient un paquetage ‘sdnaspects.placement.model’ qui représente le modèle de données spécifique à l’implantation de l’algorithme de placement basé sur la recherche de flot maximum avec un coût minimum.

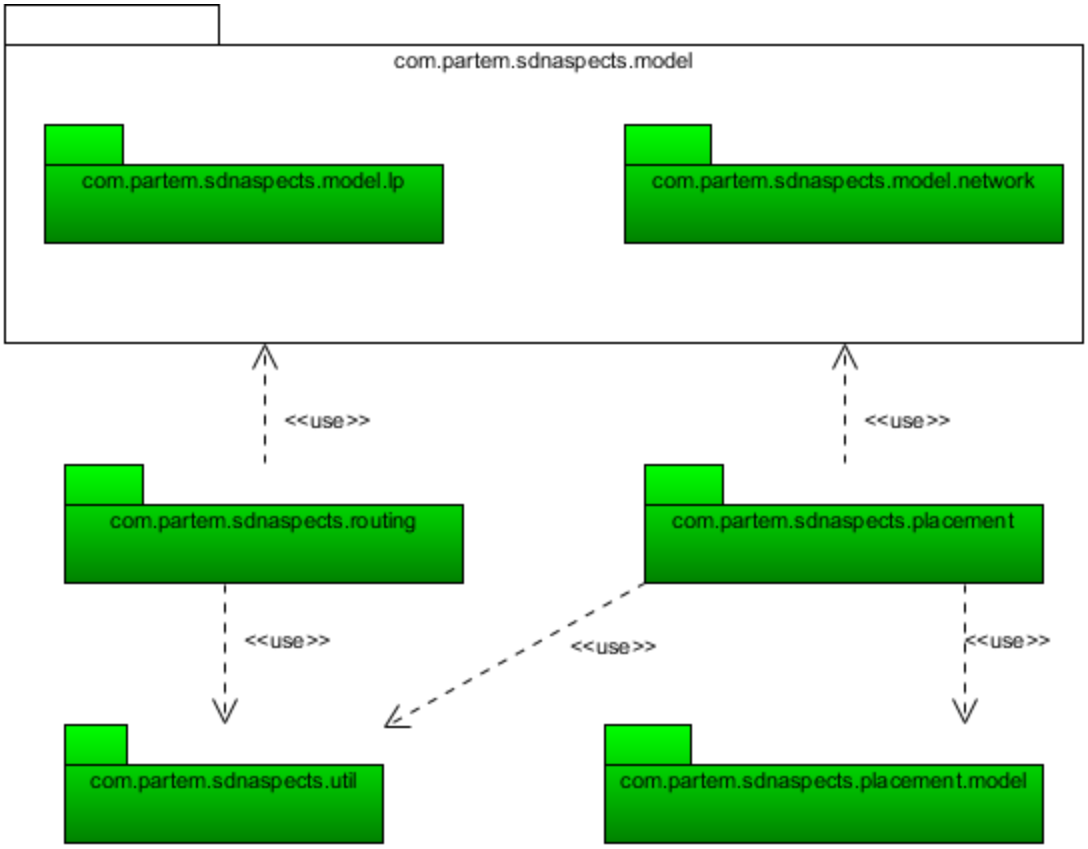


Figure 5.2 Architecture de notre outil

Conception détaillée

Dans cette partie, nous présentons la conception détaillée de quelques paquetages mentionnés dans la présentation de l’architecture.

La Figure 5.3 présente la conception détaillée du composant ‘sdnaspects.model’. Cette conception est illustrée par un diagramme de classes.

Un réseau est représenté par la classe ‘Network’ et il contient un ensemble d’équipements ‘Device’ qui sont liés entre eux par des arcs de type ‘Edge’. Un équipement peut être un commutateur ‘Switch’ ou un hôte ‘Host’.

Un programme linéaire ‘LP’ est constitué principalement d’une fonction objectif de type ‘LPExpression’. La classe ‘LPExpression’ modélise une expression de type $\sum_{i=1}^{n} c_i \; x_i$. ‘LP’ contient aussi un ensemble de contraintes de type ‘LPConstraint’. La classe ‘LPConstraint’ étend la classe ‘LPExpression’ pour pouvoir spécifier des inégalités et des égalités de la forme :

$\sum_{i=1}^{n} c_i \; x_i \; \Gamma \; a$ t.q. $\Gamma \in \{\leq, \geq, <, >, =\}$ et a une constante.

La définition de notre propre modèle de programme linéaire est motivée par notre souci d’indépendance des modèles offerts par les outils d’optimisation et la possibilité de changer d’outil facilement.

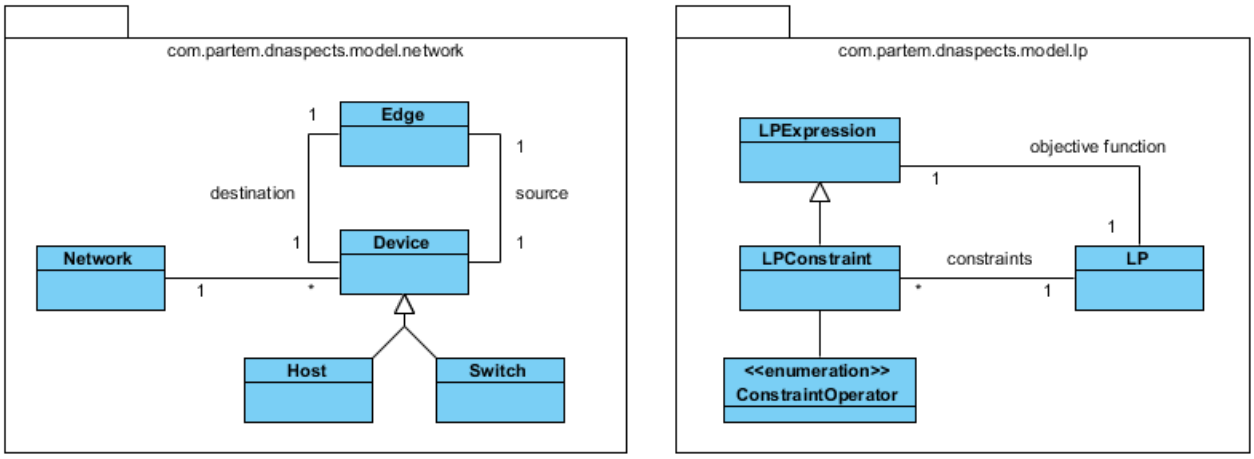


Figure 5.3 Diagramme de classes du composant ‘sdnaspects.model’

La Figure 5.4 présente le diagramme de classes du module de placement. Ce module possède aussi un modèle de données propre à lui. Ce modèle se manifeste par le paquetage ‘sdnaspects.placement.model’ et il représente le graphe de compatibilité utilisé dans l’algorithme de placement basé sur la recherche de flot maximum avec un coût minimum. Cet algorithme est implanté dans la classe ‘RulesPlacer’ et il utilise principalement le graphe de compatibilité ‘CompatibilityGraph’. Ce dernier est construit par la classe ‘CompatibilityGraphBuilder’ qui utilise principalement l’espace des entêtes ‘RulesSpace’ représentant les règles. L’espace des entêtes est construit à partir d’une liste de règles génériques par la classe ‘RulesSpaceBuilder’.

La méthode de placement de règles basé sur le MOILP est implantée principalement dans la classe 'EpsilonConstraintDichotomicScheme'. Cette classe implante notre algorithme dichotomique qui met en oeuvre la méthode epsilon contrainte. Cette méthode se base sur la résolution de plusieurs instances d'un programme linéaire en nombres entiers, d'une façon itérative, jusqu'à l'obtention de la solution désirée. Le programme linéaire est construit par la classe 'PlacementLPBuilder' qui se base aussi sur l'espace des entêtes. La résolution de ce programme est faite par un outil d'optimisation externe. De ce but, nous avons créé la classe 'OptimizationUtil' qui transforme le modèle représentant un programme linéaire en un modèle propriétaire à l'outil d'optimisation afin de le résoudre.

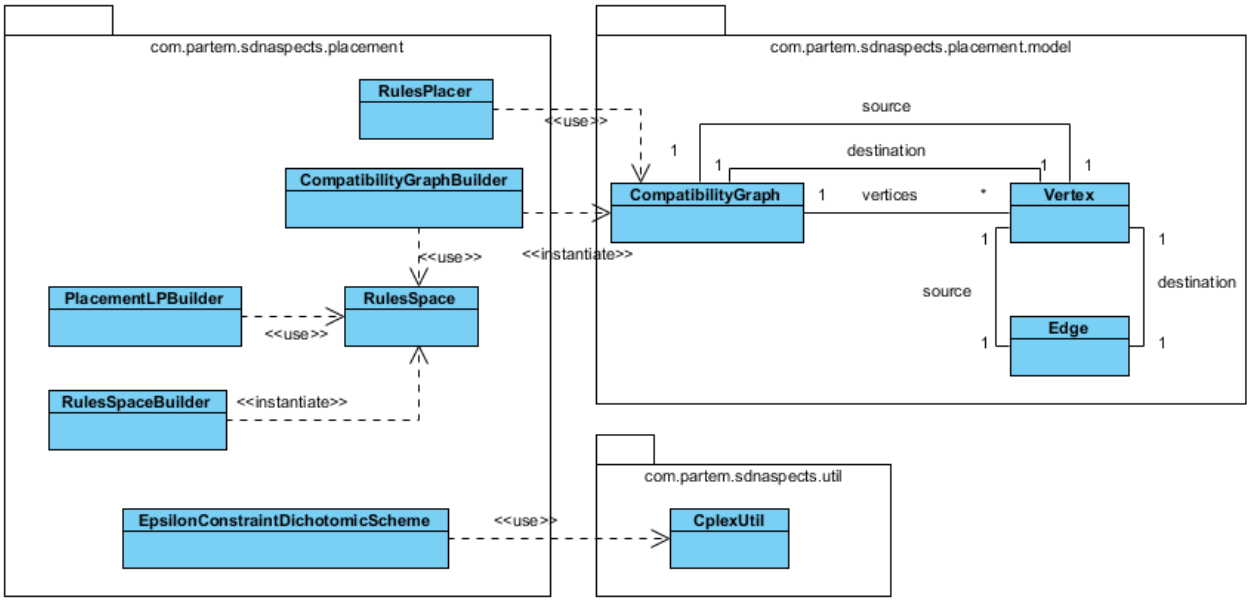


Figure 5.4 Diagramme de classes du module de placement

5.1.3 Outils utilisés

Notre outil a été implanté en utilisant le langage de programmation Java. Nous avons utilisé l'outil Maven afin d'organiser notre projet et de gérer son cycle de vie. Les tests unitaires ont été développés en se basant sur le cadriciel de test, JUnit.

Nous avons également utilisé un outil d'optimisation afin de résoudre nos programmes linéaires en nombres entiers. L'article (Meindl and Templ, 2012) présente une comparaison des outils d'optimisation linéaire. Ces outils sont classés en deux catégories, à savoir commerciaux et libres.

Les outils libres incluent :

1. GPLK (GNU Linear Programming Kit).
2. LP_SOLVE.

Les outils commerciaux incluent :

1. CPLEX.
2. XPRESS.
3. GUROBI.

Les auteurs ont testé les différents outils en utilisant un banc de test contenant 87 problèmes d'optimisation MILP. Les résultats de ces tests sont illustrés par le Tableau 5.1. La deuxième colonne indique le temps d'exécution de la totalité du banc de test tandis que la troisième colonne indique le pourcentage des problèmes résolus.

On remarque bien que les outils commerciaux sont beaucoup plus performants que les outils libres, vu qu'ils permettent de résoudre plus de problèmes en peu de temps.

Pour résoudre nos problèmes, nous avons opté pour CPLEX. Notre choix est motivé par la notoriété de CPLEX et par le fait que IBM offre une version académique gratuite afin de l'utiliser dans les recherches.

Cependant, notre conception permet de changer facilement l'outil d'optimisation étant donné le découplage entre la logique métier et les interfaces de programmations non-propriétaires .

Tableau 5.1 Résultats des tests des outils d'optimisation

Outil	temps d'exécution	% des modèles résolus
GPLK	22.11	3.45
LP_SOLVE	19.40	5.75
CPLEX	1.45	83.91
XPRESS	1.29	85.06
GUROBI	1.00	88.51

IBM ILOG CPLEX (CPLEX, 2009) est un outil propriétaire à IBM qui permet la modélisation et la résolution d'une panoplie de problèmes d'optimisation tels que :

1. Les programmes linéaires (LP)

2. Les programmes quadratiques (QP)
3. Les programmes linéaires mixtes (MILP)

CPLEX peut être intégré dans plusieurs environnements de développement moyennant la technologie **Concert**. Cette technologie consiste en un ensemble de librairies qui exposent des interfaces de programmation pour plusieurs environnements à savoir Java, C++, et .Net.

La Figure 5.5 suivante présente une architecture typique d'une application Java qui utilise CPLEX. L'interface de programmation de la technologie Concert est illustrée par les carreaux en orangé. Cette interface permet l'accès aux fonctionnalités de CPLEX à travers deux genres d'objets :

1. Objets de modélisation : Ces objets permettent la modélisation des problèmes d'optimisation. Les expressions qui servent à la modélisation des contraintes et de la fonction objectif sont de type IloNumExpr. Les variables sont de type IloNumVar.
2. Objets de résolution : Ces objets permettent la résolution des modèles d'optimisation. Les objets de résolution sont des instances de la classe IloCplex. Mise à part la résolution, cette classe permet d'accéder à l'état de la solution afin de récupérer des informations quant à sa faisabilité ou son optimalité. D'autres informations concernant la solution peuvent être également obtenues par IloCplex. Ces informations peuvent contenir les valeurs des variables ou la valeur de la fonction objectif. De plus, IloCplex permet de choisir les algorithmes d'optimisation à utiliser et de modifier leurs paramètres.

5.1.4 Algorithme glouton de placement

La complexité de l'algorithme de placement basé sur le flot maximum avec un coût minimum calculé par l'algorithme Ford-Fulkerson avec l'heuristique de plus court chemin d'augmentation est $O(\ p\ (m + n\ log(n)))$ où les variables m, n et p désignent respectivement les nombres d'arcs, des noeuds et des noeuds qui représentent les portions dans le graphe G_f. L'algorithme recherche dans le pire cas p chemins d'augmentation. Chaque chemin se calcule en utilisant l'algorithme Dijkstra qui peut se faire en $O(m + n\ log(n))$ si on utilise la structure de données Tas de Fibonacci (Fibonnacci heap).

Étant donné le nombre élevé de portions que peut générer la distribution des règles dans l'espace des entêtes, l'application de cet algorithme peut être coûteuse.

Pour réduire le coût, nous avons procédé à une modification qui transforme l'algorithme basé

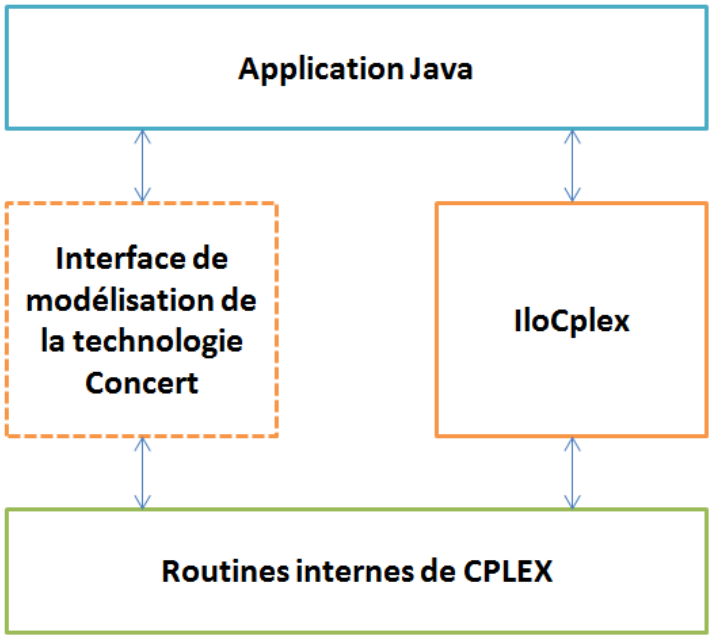


Figure 5.5 Architecture d'une application Java basée sur CPLEX

sur la recherche de 'Min Cost Max Flow' en un algorithme glouton qui se résume à placer chaque portion dans le commutateur ou on peut placer un maximum de ses soeurs.

Ce placement commence par le placement des portions ayant un nombre plus élevé de soeurs qui peuvent être placées avec elles. Pour ce faire, nous avons modifié l'algorithme de Ford-Fulkerson avec l'heuristique de plus court chemin en ignorant les chemins de déplacement. Un chemin de déplacement est un chemin d'augmentation qui se trouve seulement sur un graphe résiduel et qui permet de déplacer une portion pour libérer la place au placement d'une autre.

Soit i la portion à déplacer d'un commutateur s_k vers un autre commutateur s_l pour libérer de la place à la portion j. Ceci se ramène à annuler le flot passé sur l'arc $(i,\ s_k)$ et à passer du flot sur les arcs $(i,\ s_l)$ pour placer i dans s_l et $(j,\ s_k)$ pour placer j dans s_k. Le chemin de déplacement correspondant à cette opération est $((j,\ s_k), (s_k,\ i), (i,\ s_l))$. Ce concept sera illustré dans un exemple par la suite.

L'algorithme glouton est présenté dans la Figure 5.6. Cet algorithme commence par trier les noeuds adjacents à l'origine et qui représentent les portions. Le tri des noeuds représentant les portions se base sur le critère 5.1 qui représente le minimum des coûts des arcs issus d'une

portion i et se fait dans un ordre croissant.

$$min\{c_a : a \in A_i^B\} \tag{5.1}$$

Les plus courts chemins d'augmentation seront parcourus l'un après l'autre en partant de l'origine o et en parcourant ses noeuds adjacents (portions) triés dans l'ordre. À chaque itération, on cherche le chemin qui commence par l'arc liant l'origine o à la portion actuelle et qui atteint la destination t. Si un tel chemin existe alors on augmente le flot d'une unité. Ensuite, on passe à la recherche de plus court chemin suivant qui passe par le noeud de la portion suivante dans la liste des adjacents de o. Cette opération se répète pour chaque portion dans la liste des adjacents de l'origine.

```
procédure approximationGlouton( Gf  g ){
        trierLaListeDesAdjacentsA( g.o ,
                        minDesCoutDesArcsIssusDeChaqueElementDeLaListe);
        portionAPlacer=0;
        tantque(existeUnCheminDAugmentationQuiPassePar(portionAPlacer)
        || portionAPlacer < nbPortions){
                P = cheminDAugmentation;
                si(P!=Nil){
                        augmenterLeFlot(P, 1);
                }
                portionAPlacer++;
        }
}
```

Figure 5.6 Algorithme glouton

Par la suite, nous présentons l'exécution de l'algorithme de placement basé sur le calcul de flot maximum avec un coût minimum et l'algorithme glouton sur le même exemple de graphe de compatibilité G_f. Cet exemple concerne le placement de 3 portions de règles, à savoir p_{11}, p_{12} et p_{21}. Les portions p_{11} et p_{12} appartiennent à la même règle r_1 et la portion p_{21} appartient à la règle r_2. La portion p_{11} peut être placée dans le commutateur s_2 alors que la portion p_{12} peut être placée dans le commutateur s_1. La portion p_{21} peut être placée dans les commutateurs s_1 et s_2. La capacité du commutateur s_1 est égale à 2 et la capacité de s_2 est égale à 1. La Figure 5.7 présente le graphe G_f de notre exemple. Chaque arc est annoté par le couple $u_a,\ c_a$ dénotant respectivement la capacité et le coût associés à cet arc.

La Figure 5.8 illustre un exemple d'exécution de l'algorithme de Ford-Fulkerson basé sur l'heuristique de plus court chemin d'augmentation.

Le graphe de la Figure 5.8a montre l'initialisation des fonctions potentielles qui serviront au calcul des coûts réduits afin d'éviter d'avoir des coûts négatifs vu qu'on utilise Dijkstra pour le calcul des plus courts chemins d'augmentation. La Figure 5.8b montre la recherche du premier plus court chemin d'augmentation (en noir) qui a permis de placer la portion p_{21} dans le commutateur s_2. Chaque noeud n est annoté par la distance $d(n)$ utilisée dans le calcul de plus court chemin par Dijsktra. La Figure 5.8c illustre le graphe résiduel qui montre que les arcs $(o,\ p_{21})$ et $(s_2,\ t)$ sont déjà saturés. Les annotations de type f_a/u_a indiquent le flot f_a qui passe par un arc a de capacité u_a. Si le flot f_a est inférieur à la capacité u_a, alors il est possible de passer davantage de flot sur l'arc a et de passer le flot f_a à travers l'arc opposé ajouté au graphe résiduel. L'arc opposé est désigné par une flèche qui va dans l'autre sens de l'arc original. De ce fait, les flèches associées aux arcs des figures indiquent les sens où on peut passer du flot. La Figure 5.8d montre le calcul du deuxième plus court chemin d'augmentation (en noir) qui permet de placer la portion p_{12} dans le commutateur s_1. La Figure 5.8e montre que l'arc $(o,\ p_{12})$ est devenu saturé tandis que l'arc $(s_1,\ t)$ peut encore faire passer une unité de flot. La Figure 5.8f montre la recherche du dernier plus court chemin d'augmentation (en noir). Ce chemin est un chemin de déplacement qui annule le flot dans l'arc $(p_{21},\ s_2)$ en faisant passer du flot dans le sens contraire. Ce chemin permet de déplacer la portion p_{21} du commutateur s_2 vers le commutateur s_1 afin de pouvoir placer la portion p_{11} dans le commutateur s_2. Finalement, l'algorithme a permis le placement des portions p_{12} et p_{21} dans le commutateur s_1 et le placement de la portion p_{11} dans le commutateur s_2.

Le chemin de déplacement (en noir) illustré par la Figure 5.8f est obtenu dans le cas ou un graphe résiduel est utilisé pour avoir la possibilité d'annuler le flot sur un arc donné comme on l'a fait pour l'arc $(p_{21},\ s_2)$. Cette opération permet de déplacer une portion qui peut être placée ailleurs pour laisser la place à une autre portion. On ne permet le déplacement d'une portion uniquement s'il induit la libération de l'espace sur des commutateurs saturés. Ceci se réalise si et seulement si la portion à déplacer n'a pas de portions soeurs dans le même commutateur où elle a été placé.

La Figure 5.9 illustre un exemple d'exécution de l'algorithme glouton sur le graphe G_f de la Figure 5.7. La Figure 5.9a illustre la première étape qui sert à trier les portions selon le minimum des coûts associés à leurs arcs menant vers les commutateurs. Les coûts minimums représentant les critères du tri sont désignés par la couleur orange. La Figure 5.9b illustre deux chemins d'augmentation. Le premier chemin (en noir) a été trouvé dans la 1re itération alors que le deuxième chemin (en vert) a été trouvé dans la 3e itération. En effet, dans la 1re

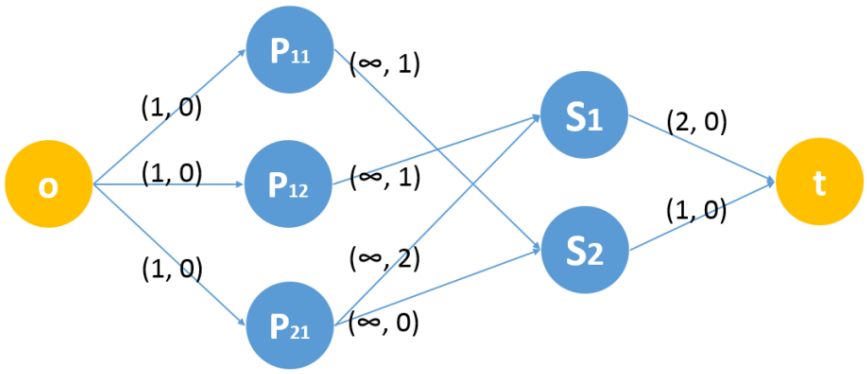
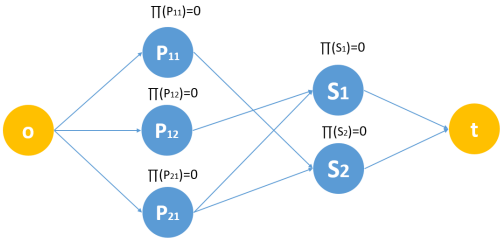


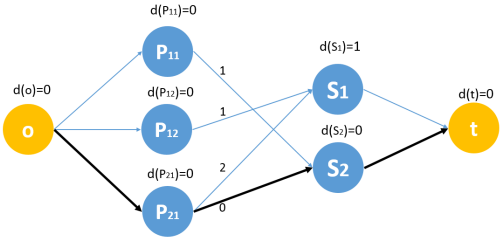
Figure 5.7 Exemple d'un graphe G_f avec les capacités et les coûts initiaux

itération l'algorithme essaie de trouver un chemin d'augmentation qui commence par l'arc $(o,\ p_{21})$. L'augmentation de flot à travers ce chemin (en noir) sature l'arc $(s_2,\ t)$. Dans la deuxième itération, l'algorithme essaie de trouver un chemin d'augmentation qui commence par l'arc $(o,\ p_{11})$. Ceci n'est pas possible vu que le seul chemin qui mène vers t et qui commence par l'arc $(o,\ p_{11})$ passe par l'arc $(s_2,\ t)$ qui a été déjà saturé dans la 1re itération. Finalement, dans la 3e itération l'algorithme essaie de trouver un chemin d'augmentation qui commence par l'arc $(o,\ p_{12})$. Le chemin trouvé est désigné en vert. La Figure 5.9c montre le flot passé dans les arcs de G_f. Cet algorithme a permis le placement des portions p_{21} et p_{12} respectivement dans les commutateurs s_2 et s_1.

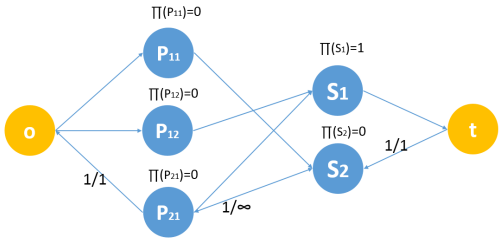
La complexité de l'algorithme glouton est de $O(\ p\ log(p) + p\ (\ 2\ +\ lp))$ où lp désigne la longueur de plus long chemin de placement dans le réseau. $O(p\ log(p))$ désigne la complexité de l'opération de tri des noeuds adjacents au noeud origine en utilisant le tri par fusion. Chaque chemin d'augmentation est formé par 3 arcs (1- de l'origine vers la portion ; 2- de la portion vers le commutateur ; 3-du commutateur vers la destination). Ce chemin est déterminé en $O(2\ +\ lp)$. En effet, le premier et le dernier arc du chemin sont déterminés en deux opérations alors que le deuxième arc est déterminé au pire des cas en lp comparaisons puisque on doit choisir l'arc le moins coûteux qui mène vert l'un des commutateurs situés dans le chemin de placement de la portion considérée. De ce fait, on aura à choisir entre lp arcs qui mènent vers lp commutateurs dans le pire des cas.



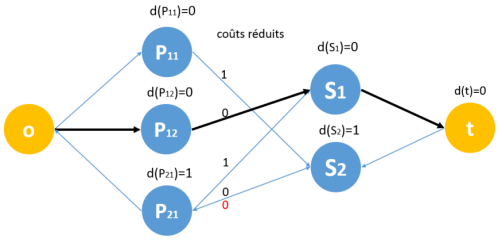
(a) Initialisation des fonctions potentielles



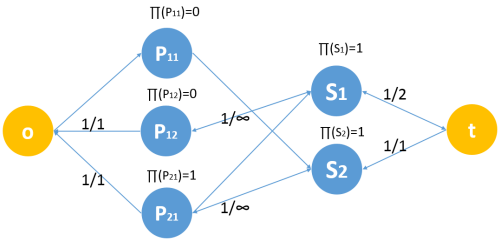
(b) Calcul de plus court chemin sur le graphe résiduel en se basant sur les coûts réduits



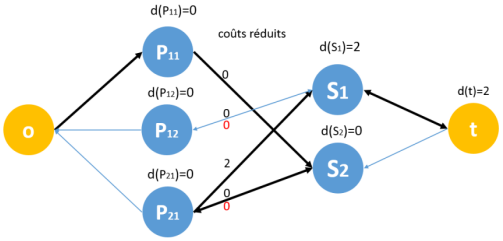
(c) Graphe résiduel montrant l'augmentation du flot à travers le chemin d'augmentation trouvé dans l'étape précédente et la mise à jour des fonctions potentielles



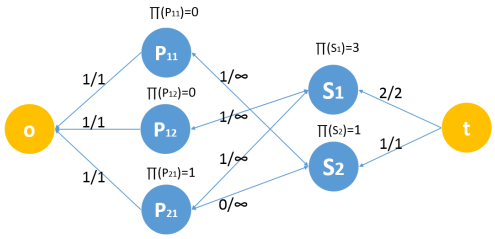
(d) Calcul de plus court chemin sur le graphe résiduel en se basant sur les coûts réduits



(e) Graphe résiduel montrant l'augmentation du flot à travers le chemin d'augmentation trouvé dans l'étape précédente et la mise à jour des fonctions potentielles

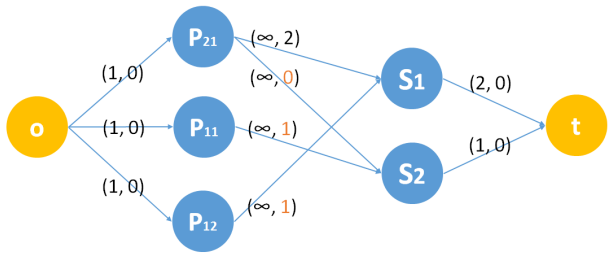


(f) Calcul de plus court chemin sur le graphe résiduel en se basant sur les coûts réduits

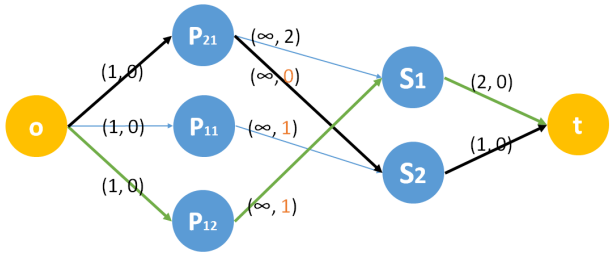


(g) Graphe résiduel montrant l'augmentation du flot à travers le chemin d'augmentation trouvé dans l'étape précédente et la mise à jour des fonctions potentielles

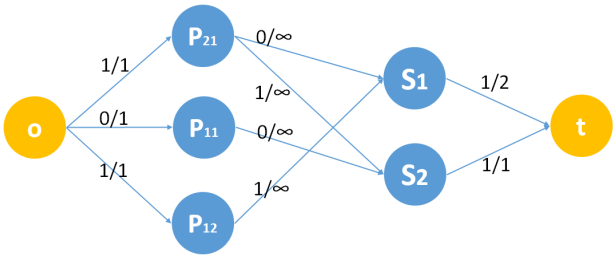
Figure 5.8 Exécution de l'algorithme Ford-Fulkerson avec l'heuristique de plus court chemin



(a) Tri des portions



(b) Les deux chemins d'augmentation



(c) Résultat final contenant le flot passé dans chaque arc

Figure 5.9 Exécution de l'algorithme glouton

5.2 Tests et résultats

5.2.1 Banc de test

Afin de tester la rectitude et les performances de notre outil, nous avons besoin des éléments suivants :

1. Un générateur de graphes modélisant des topologies de réseaux.
2. Un générateur de requis modélisant des contraintes sur les bandes passantes et des politiques de compositions des points d'acheminement.
3. Un générateur de règles à base du caractère générique '*'. Ces règles vont servir comme entrée aux approches de placement.

Afin d'essayer nos approches sur des infrastructures réalistes, nous avons eu recours à l'outil GT-ITM (Georgia Tech Internetwork Topology Models). Cet outil se base sur les techniques de l'article (Zegura et al., 1996), afin de générer des graphes représentant des topologies qui ressemblent à de vrais réseaux. GT-ITM génère des graphes, principalement, sous le format SGB de Don Knuth. Il offre, également, d'autres formats plus faciles à analyser et à transformer en notre modèle représentant un réseau. La Figure 5.10 représente un exemple de réseau dont la topologie est générée par GT-ITM et qui contient 25 commutateurs (les cercles) et 21 hôtes (les rectangles) dont les adresses sont générées par notre outil. Le réseau généré est de type 'transit-stub' et il contient 3 sous réseau connectés par un commutateur central. Les hôtes appartenant à chaque sous-réseau ont les mêmes préfixes d'adresses qui sont représentés sur 8 bits dans notre exemple. Après avoir construit le réseau à tester et adressé les hôtes,

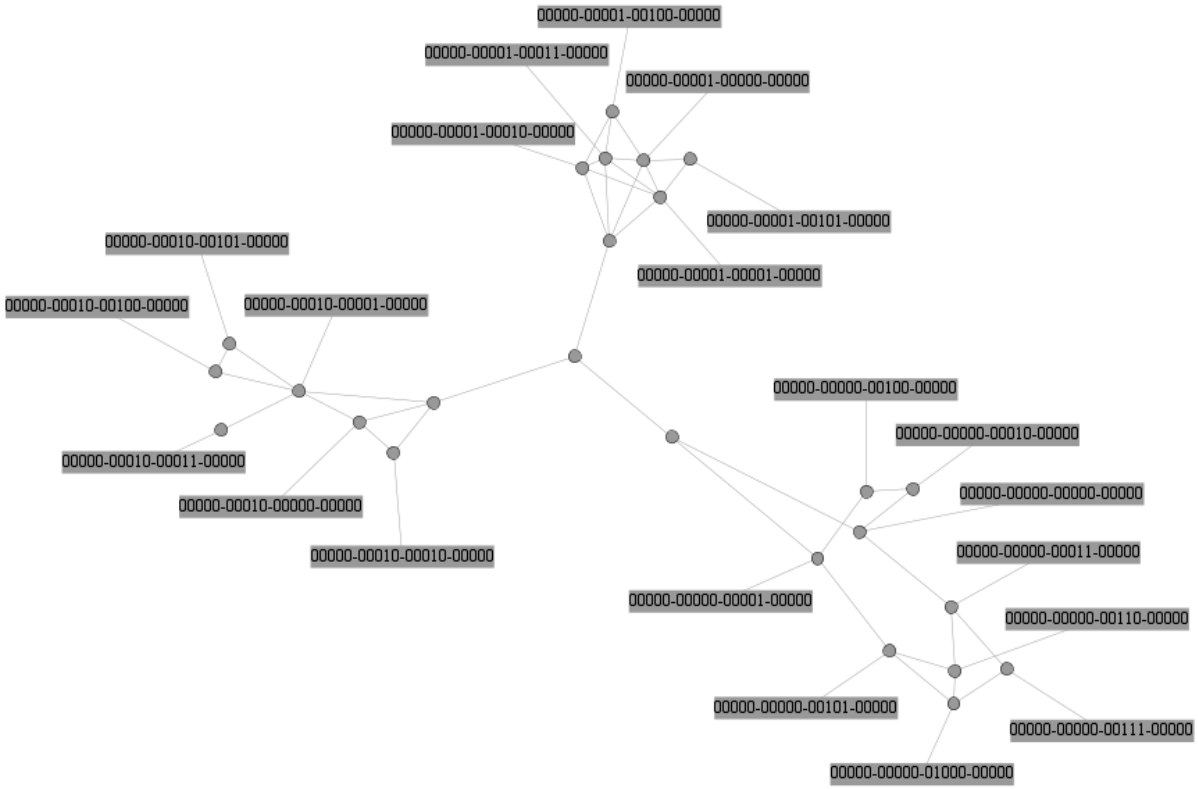


Figure 5.10 Exemple de réseau généré par l'outil GT-ITM

nous procédons à la génération des requis et des règles qui seront utilisés respectivement par les modules de routage et de placement. Les requis destinés au test du module de routage sont générés d'une façon aléatoire. Chaque requis représente un flux caractérisé par une source et une destination fixes et par d'autres caractéristiques telles que le protocole utilisé et le type d'application. Afin de simplifier la génération des requis, nous représentons les autres

caractéristiques par un simple id. De plus, pour chaque requis, nous générons des contraintes de la garantie et la limitation de la bande passante et nous générons aussi une politique de composition des points d'acheminement. D'autre part, les règles destinées au test du module de placement sont plus compliquées à générer. La Figure 5.11 présente un ensemble de règles générées par notre outil pour le réseau de la Figure 5.10. Pour chaque règle, nous présentons seulement le domaine d'application qui est formé en l'occurrence de 4 champs, à savoir l'adresse de la source (srcAdr), la destination de la source (dstAdr), le numéro du port de la source (srcPort) et le numéro du port de la destination (dstPort). La génération des adresses génériques est faite en parcourant aléatoirement un arbre représentant tous les préfixes des adresses des hôtes. La Figure 5.12 représente une partie de l'arbre d'adressage relatif au réseau de la Figure 5.10. Le parcours du chemin bleu génère l'adresse générique suivante : '00000-00001-00000-*****'

```
Rule01::: srcAdr: 0000000001xxxxxxxxx dstAdr: 00000000000100000000 srcPort: 32 dstPort: 39
Rule02::: srcAdr: 0000000001xxxxxxxxx dstAdr: 00000xxxxxxxxxxxxxx srcPort: 96 dstPort: 24
Rule03::: srcAdr: 000000001000010xxxxx dstAdr: 000000001000010xxxxx srcPort: 97 dstPort: 22
Rule04::: srcAdr: 00000xxxxxxxxxxxxxx dstAdr: 000000001000010xxxxx srcPort: 79 dstPort: 1
Rule05::: srcAdr: 00000xxxxxxxxxxxxxx dstAdr: 00000000010001000000 srcPort: 45 dstPort: 63
Rule06::: srcAdr: 00000xxxxxxxxxxxxxx dstAdr: 00000000100000000000 srcPort: 74 dstPort: 67
Rule07::: srcAdr: 00000xxxxxxxxxxxxxx dstAdr: 000000000100100xxxxx srcPort: 18 dstPort: 47
Rule08::: srcAdr: 0000000000xxxxxxxxx dstAdr: 00000xxxxxxxxxxxxxx srcPort: 48 dstPort: 19
Rule09::: srcAdr: 00000xxxxxxxxxxxxxx dstAdr: 00000000000000100000 srcPort: 41 dstPort: 97
Rule10::: srcAdr: 00000xxxxxxxxxxxxxx dstAdr: 000000000001000xxxxx srcPort: 51 dstPort: 4
Rule11::: srcAdr: 00000xxxxxxxxxxxxxx dstAdr: 0000000001xxxxxxxxx srcPort: 40 dstPort: 81
Rule12::: srcAdr: 0000000001xxxxxxxxx dstAdr: 000000000000001xxxxx srcPort: 36 dstPort: 48
```

Figure 5.11 Exemple d'une liste de règles génériques

5.2.2 Tests de rectitude

La vérification de la rectitude des solutions générées est effectuée par des tests unitaires développés pour mettre en oeuvre la technique TDD.

Ces tests unitaires permettent l'automatisation de la vérification vu que nous considérons de grands exemples dont la rectitude ne peut pas être vérifiée manuellement. En effet, ces tests seront roulés après la génération de chaque solution.

La vérification de la rectitude de chaque routage est basée sur la vérification de l'accessibilité entre la source et la destination concernées par le routage. De ce fait, nous prenons le routage

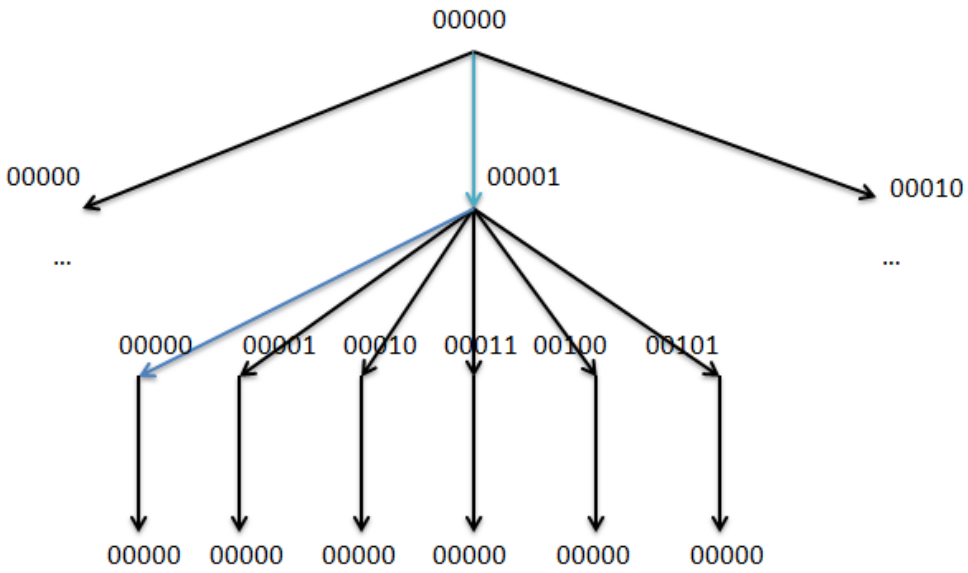


Figure 5.12 Une partie de l'arbre d'adressage relatif à notre exemple

généré pour chaque requis et nous vérifions si ce routage mène de la source vers la destination spécifiée.

De plus, pour les requis spécifiant une politique de composition de points d'acheminement, nous vérifions si le routage passe par chaque niveau de points d'acheminement, dans l'ordre.

Pour les requis spécifiant des contraintes de garantie et de limitation de la bande passante, nous vérifions les deux propriétés suivantes :

1. La capacité réservée à la garantie de la bande passante sur un lien doit être supérieure ou égale à la somme des bandes passantes garanties des flux acheminés à travers ce lien.
2. Pour un lien donné, la capacité maximale réservée à la limitation de la bande passante doit être supérieure ou égale à la bande passante limite de chaque flux acheminé à travers ce lien.

Finalement, nous vérifions que l'ensemble des flux acheminés à partir d'un commutateur donné ne doit pas dépasser la capacité de ce commutateur.

Dans le volet de placement des règles, nous vérifions les propriétés suivantes :

1. Chaque portion doit être placée dans un seul commutateur.
2. Le nombre des règles placées dans un commutateur donné ne doit pas dépasser la capacité de ce commutateur.

5.2.3 Performances

Les mesures de performances ont été effectuées sur un ordinateur équipé d'un processeur i7 avec 6GB de RAM.

Dans un premier temps, nous avons testé les performances du programme linéaire en nombres entiers qui permet de mettre en oeuvre la garantie et la limitation de la bande passante ainsi que les politiques de composition des points d'acheminement. Ce programme a été testé en utilisant deux topologies générées par GT-ITM.

La première topologie présentée à la Figure 5-9 contient 25 commutateurs et 21 hôtes. La deuxième topologie est composée de 100 commutateurs et de 84 hôtes.

La politique de routage existante a été générée moyennant l'algorithme de Dijkstra en se basant sur des coûts générés aléatoirement par GT-ITM.

Les résultats des tests effectués sur les deux topologies sont illustrés dans les Figures 5.13 et 5.14. L'axe des abscisses représente le nombre de flux pour lesquelles on cherche un routage qui met en oeuvre les aspects considérés et l'axe des ordonnées représente le temps d'exécution en seconde.

Pour chaque cas de test, nous imposons à tous les flux des contraintes de garantie et de limitation de la bande passante. De plus, nous imposons à 10% de ces flux de suivre des politiques de compositions générées aléatoirement.

Le temps d'exécution pour la première topologie est de l'ordre de quelques secondes pour un nombre de flux allant de 10 à 10000. Ce temps peut atteindre une demi-heure si nous cherchons les routages correspondants à 24000 flux.

Les résultats qui concernent la deuxième topologie montrent que le temps d'exécution varie de quelques secondes pour 4000 flux jusqu'à 1 heure 20 minutes pour 13300 flux.

La RAM disponible sur notre ordinateur n'a pas permis d'aller au-delà de 13300 flux pour la deuxième topologie vu que la résolution d'un grand programme linéaire en nombres entiers nécessite une grande RAM.

Dans une deuxième étape, nous avons testé les trois approches de placement que nous avons développées afin de comparer leurs performances. La première approche effectue le placement moyennant un programme linéaire multiobjectif en nombres entiers. La deuxième approche se base sur un algorithme de recherche d'un flot maximum avec un coût minimum (Min Cost Max Flow). Cet algorithme consiste en Ford-Fulkerson avec l'heuristique de plus court chemin d'augmentation. Finalement, la troisième approche consiste en une modification de

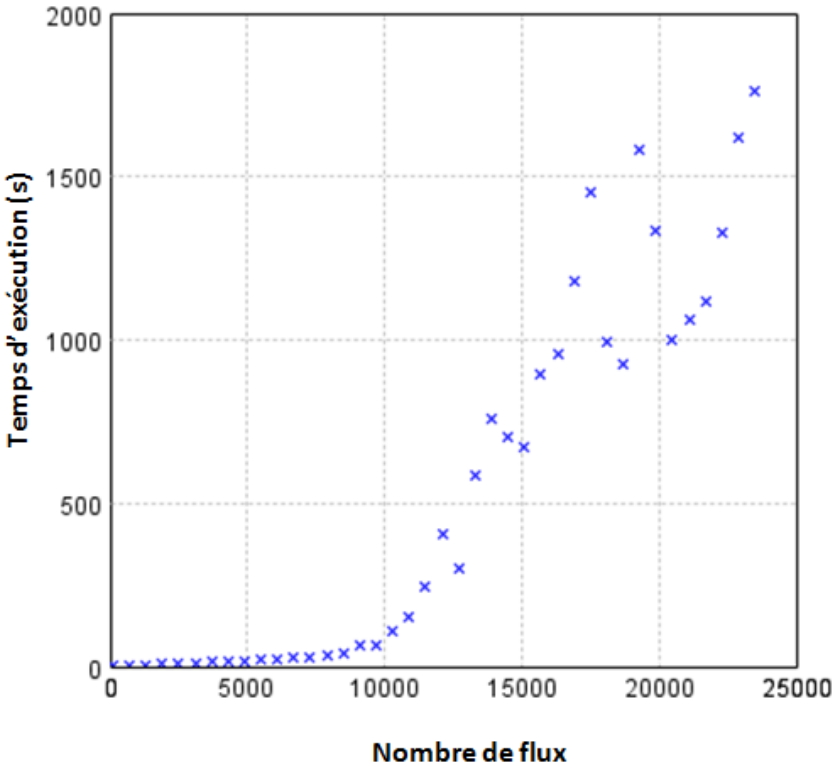


Figure 5.13 Résultats de la recherche des routages pour la première topologie

la deuxième approche qui a donné naissance à un algorithme glouton de placement.

La comparaison des performances des trois approches illustrée par la Figure 5.15 est basée sur les trois critères suivants :

1. Le temps d'exécution : Ce critère est très important vu que notre but est de réduire le temps d'exécution d'une approche à l'autre tout en gardant à peu près les mêmes performances.
2. La capacité consommée : Ce critère définit la qualité des solutions obtenues, car l'objectif est de minimiser la capacité consommée pour placer les règles.
3. Le nombre de portions placées : Ce critère définit aussi la qualité des solutions obtenues, car nous visons à maximiser le nombre de portions placées.

Les tests ont été effectués en utilisant trois topologies à savoir, topologie 1, topologie 2 et topologie 3 composées respectivement de 25 commutateurs et 23 hôtes, 100 commutateurs et 86 hôtes et 600 commutateurs et 570 hôtes. La politique de routage qui permet de définir les chemins de placement a été générée par l'algorithme de parcours en profondeur. Pour

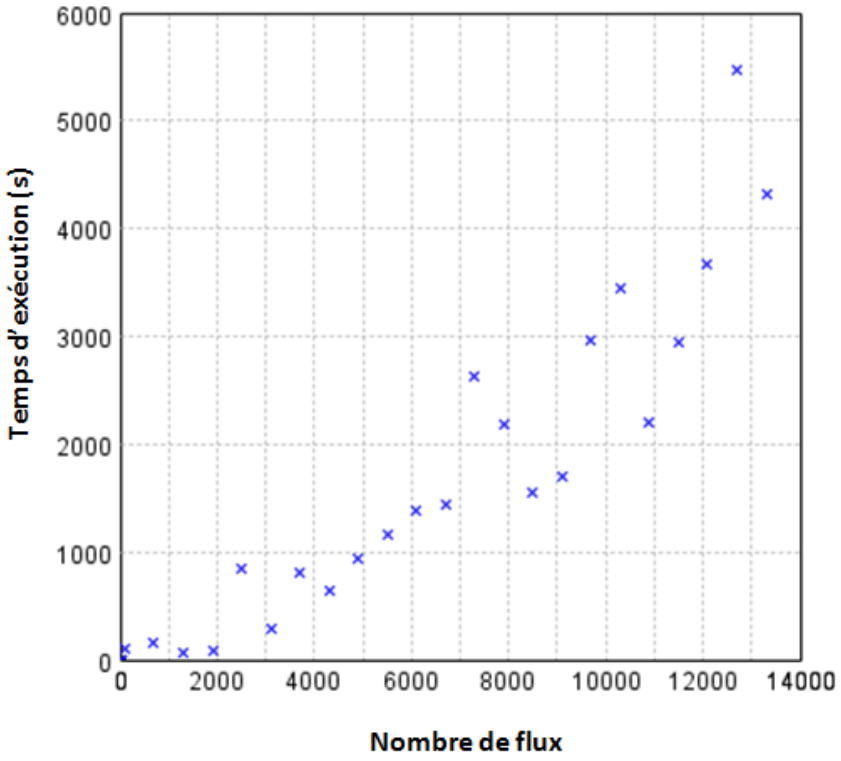
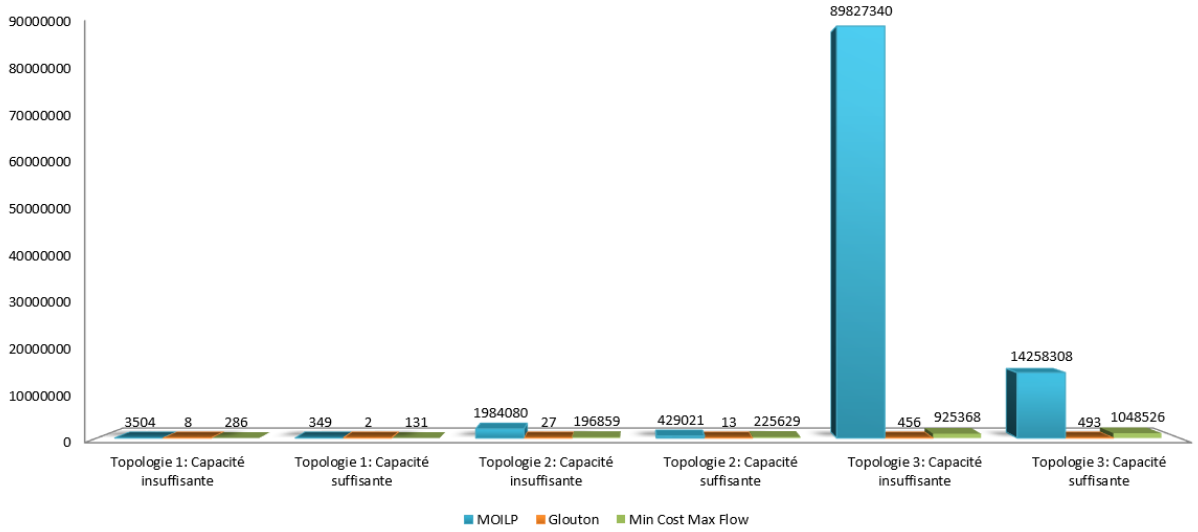
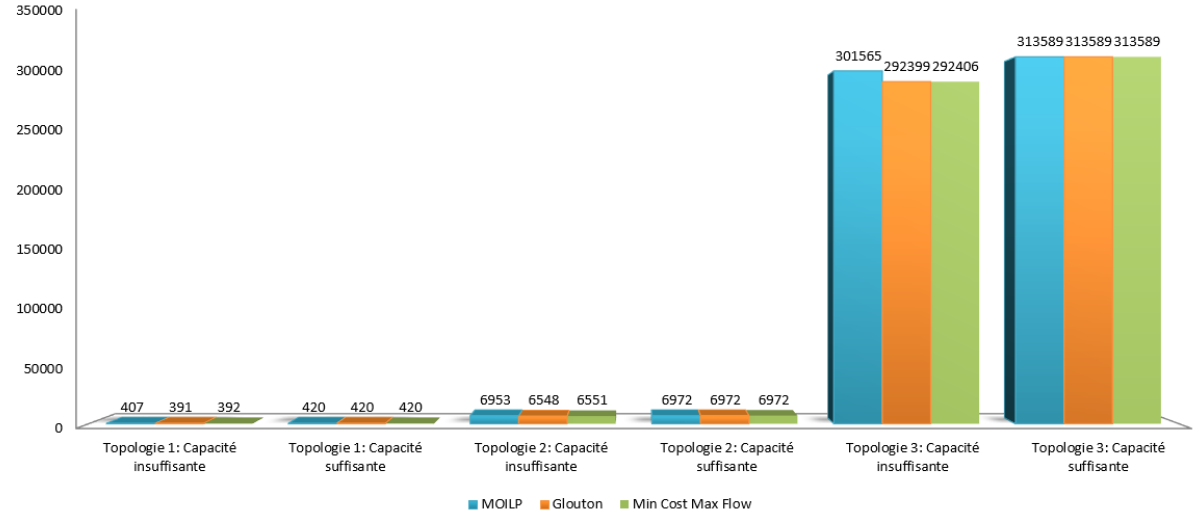


Figure 5.14 Résultats de la recherche des routages pour la deuxième topologie

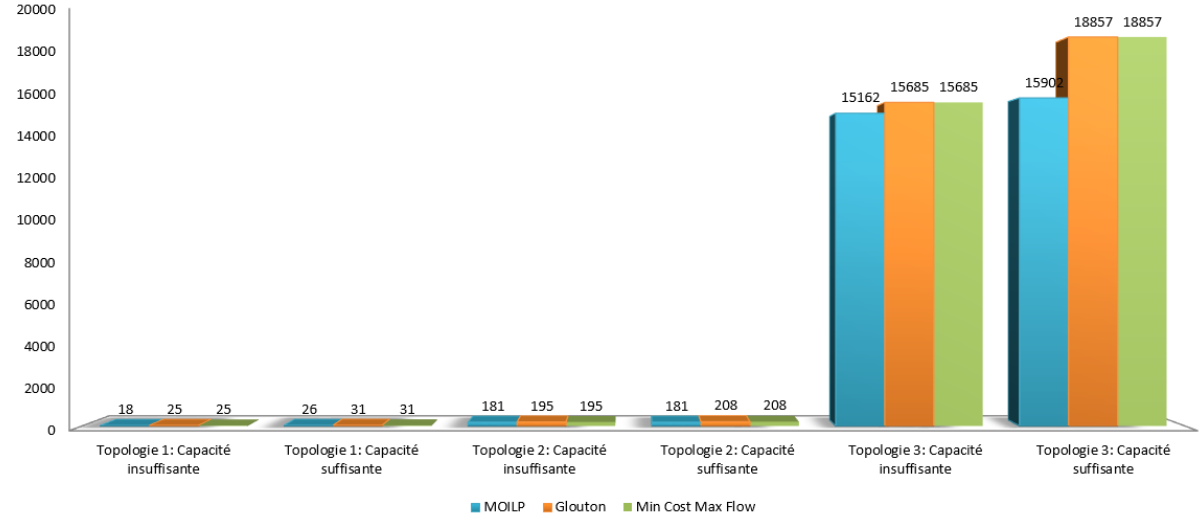
chaque topologie, nous avons effectué deux types de tests pour les trois approches. Dans le premier type de test, nous avons considéré des commutateurs avec de petites capacités afin d'observer le comportement des 3 approches dans le cas où il n'y a pas assez de capacité pour placer toutes les règles. Dans le deuxième type de test, nous avons considéré des commutateurs avec de grandes capacités afin d'observer le comportement des 3 approches dans le cas où nous pouvons placer toutes les règles. Pour chaque cas de test impliquant les 3 approches, on génère une liste de règles définies sur deux dimensions, à savoir l'adresse de la source et l'adresse de la destination. De ce fait, la distribution de ces règles dans l'espace des entêtes génère un maximum de $h^{\,2}$ portions à placer où h indique le nombre d'hôtes présents dans la topologie sous test. La Figure 5.15 illustre les résultats relatifs à l'approche utilisant un programme linéaire multiobjectif en nombres entiers (MOILP) par la couleur bleue. Les résultats de l'approche basée sur 'Min Cost Max Flow' et de l'approche basée sur un algorithme glouton sont illustrés respectivement en vert et en orange. La Figure 5.15a montre le temps d'exécution mesuré en milliseconde. Nous remarquons une différence significative entre le temps d'exécution du 'MOILP' et celui des autres algorithmes. Le temps d'exécution du 'MOILP' est important, notamment, dans les cas de tests où la capacité disponible est insuffisante pour placer toutes les règles. Ceci est dû au fait que l'algorithme $\varepsilon - contrainte$,



(a) Comparaison du temps d'exécution



(b) Comparaison des portions placées



(c) Comparaison de la capacité consommée

Figure 5.15 Comparaison des approches de placement

utilisé dans la mise en oeuvre du 'MOILP', procède à la résolution de plusieurs programmes linéaires en nombres entiers quand il ne parvient pas à placer toutes les portions dès la première itération. Le gain en terme de temps d'exécution de l'algorithme glouton par rapport à l'MOILP varie entre 99.77% et 99.99% dans les cas des tests avec capacité insuffisante. Le gain de l'algorithme 'Min Cost Max Flow' par rapport à 'MOILP' varie entre 90.07% et 98.96%. Tout en restant dans les cas des tests avec capacité insuffisante, le gain en terme de temps d'exécution de l'algorithme glouton par rapport à l'algorithme 'Min Cost Max Flow' varie entre 97.20% et 99.98%.

Dans le cas de tests avec une capacité suffisante, le gain en terme de temps d'exécution de l'algorithme 'Min Cost Max Flow' par rapport à l'MOILP varie entre 47.40% et 92.64%. Le gain de l'algorithme glouton par rapport à l'MOILP varie entre 99.42% et 99.99%. Finalement, le gain en terme de temps d'exécution de l'algorithme glouton par rapport à l'algorithme 'Min Cost Max Flow' varie entre 98.47% et 99.99%.

L'amélioration en terme de temps d'exécution d'une approche de placement à une autre est significative.

La Figure 5.15b illustre le nombre de portions placées dans chaque cas de test. Dans le cas où la capacité est suffisante, tous les algorithmes parviennent à placer le même nombre de portions. Par exemple, les trois approches ont permis le placement de toutes les portions (6972) dans le cas de test de la topologie 2 avec une capacité suffisante.

Dans le cas où la capacité est insuffisante, nous remarquons que le MOILP permet de placer un peu plus de portions que les autres algorithmes. En effet, le gain en terme de nombre de portions placées de l'MOILP par rapport à l'algorithme basé sur 'Min Cost Max Flow' varie entre 3.03% et 5.78%. Le gain de l'algorithme 'Min Cost Max Flow' par rapport à l'algorithme glouton varie entre 0.002% et 0.25%. Ceci implique que nous n'avons pas beaucoup de chemins de déplacement qui permettent d'augmenter le flot dans l'algorithme Ford-Fulkerson avec l'heuristique de plus court chemin.

La Figure 5.15c illustre la capacité consommée dans chaque cas de test. Nous remarquons que l'algorithme glouton et l'algorithme 'Min Cost Max Flow' consomment la même capacité pour tous les cas de test. Le MOILP consomme moins de capacité que les autres algorithmes pour les deux types de cas de test. Dans le cas des tests avec capacité suffisante, le gain en terme de capacité consommée de l'MOILP par rapport aux autres algorithmes varie entre 12.98% et 16.20%.

En résumé, l'algorithme glouton permet de gagner énormément en terme de temps d'exécution par rapport aux autres approches tout en gardant presque les mêmes performances

par rapport aux celles de l'algorithme 'Min Cost Max Flow' et des performances légèrement inférieures par rapport au MOILP.

Dans une dernière étape, nous avons mesuré les performances de l'algorithme glouton avec de plus grands exemples. Les résultats de ces mesures sont illustrés dans la Figure 5.16. Les règles générées pour chaque cas de test sont définies sur 4 dimensions, à savoir l'adresse de la source, l'adresse de la destination, le numéro de port dans la source et le numéro de port dans la destination. La distribution de ces règles dans l'espace des entêtes peut générer un maximum de $p^2 h^2$ portions à placer où p est égale au nombre de ports utilisés dans les règles.

Pour chaque cas de test, nous spécifions le nombre de règles à placer et le nombre de portions qui résultent de la distribution de ces règles dans l'espace des entêtes. À titre d'exemple, le premier cas de test (Test 1) vise à placer 10000 règles qui résultent en 501019 portions.

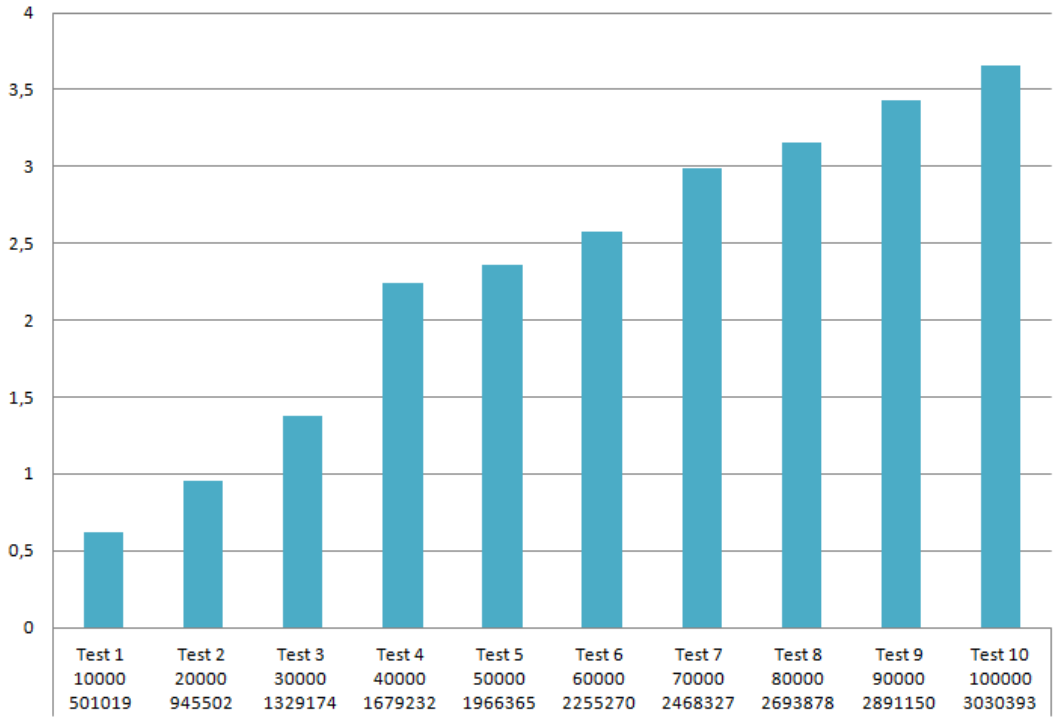
Le nombre de règles à placer varie de 10000 à 100000 et le nombre de portions varie de 501019 à 3030393.

La Figure 5.16a montre le temps d'exécution en second pour chaque cas de test. Ce temps varie entre 0.6 et 3.6 secondes.

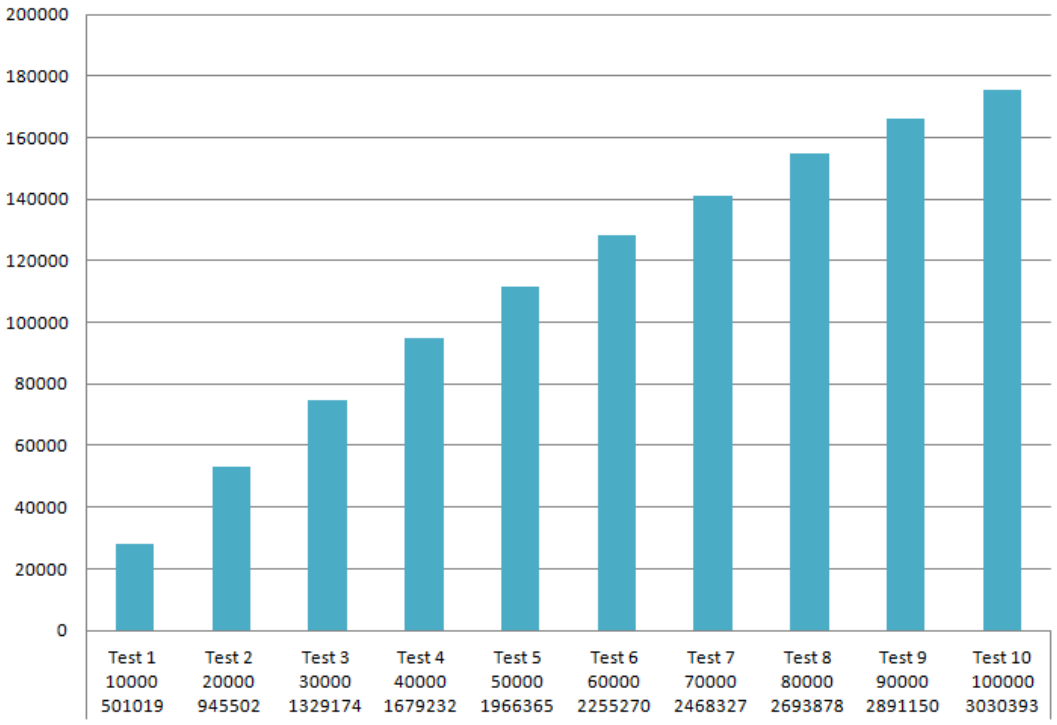
La Figure 5.16b montre la capacité consommée pour chaque cas de test. Nous remarquons que la capacité consommée pour chaque liste de règles est égale en moyenne à 2.22 fois le nombre de règles contenues dans chaque liste.

5.3 Conclusion

Ce chapitre nous a permis de présenter l'implantation de notre outil et de mesurer les performances des approches que nous avons développées. Dans un autre volet, nous avons présenté une autre approche de placement qui permet de gagner énormément en terme de temps d'exécution tout en gardant des performances compétitives.



(a) Temps d'exécution en seconde



(b) Capacité consommée

Figure 5.16 Résultats d'exécution de l'algorithme de placement glouton avec des règles définies sur 4 dimensions

CHAPITRE 6 CONCLUSION

6.1 Synthèse des travaux

Au cours de ce mémoire, nous avons élaboré le sujet de la mise en oeuvre des aspects de gestion des réseaux définis par logiciel. Nous avons considéré les trois aspects suivants :

1. la garantie et la limitation de la bande passante,
2. la composition des points d'acheminement
3. le placement des règles

La mise en oeuvre des deux premiers aspects a été basée sur un programme linéaire en nombres entiers. Ce programme cherche le routage nécessaire qui respecte les requis relatifs à ces aspects. Contrairement aux travaux existants, notre programme prend en compte les ressources disponibles, à savoir les capacités des commutateurs.

La mise en oeuvre du troisième aspect a été faite de plusieurs façons. Dans un premier temps, nous avons créé un programme linéaire multiobjectif en nombres entiers. Ce programme permet de maximiser les règles placées cependant il consomme beaucoup de temps. De ce fait, on a créé une autre approche de placement qui se base sur le problème de recherche du flot maximum avec un coût minimum "Min Cost Max Flow". Cette approche nous a permis de gagner du temps par rapport à l'approche précédente avec une petite perte en terme de performances qui se mesurent principalement par la capacité consommée et le nombre de portions placées.

Malgré le fait qu'on a gagné du temps par rapport à la première approche, nous avons créé une troisième approche encore plus rapide que la deuxième et qui garde presque les mêmes performances. Cette approche consiste en un algorithme glouton qui résulte de la modification de l'approche basée sur la recherche du flot maximum avec un cout minimum.

Nos trois approches de placement permettent de maximiser le nombre de portions placées dans le cas où on n'a pas assez de capacité dans les commutateurs de réseau considéré. À notre connaissance, la maximisation de placement des règles génériques n'a pas été considérée en aucun autre travail.

De plus, la modélisation de problème de placement est assez générique pour s'adapter à plusieurs autres cas. Nous traitons essentiellement le placement des règles génériques sans tenir compte de leurs actions. Dans le cas où l'action est importante, il suffit de considérer que les règles ayant les mêmes actions représentent une seule règle.

Dans d'autres mesures, notre modélisation peut s'adapter au cas où on ne peut installer des règles que dans certains commutateurs de réseau. Ceci s'illustre dans les réseaux hétérogènes contenant à la fois des commutateurs compatibles OpenFlow et d'autres équipements. De ce fait, il suffit de considérer seulement les commutateurs compatibles OpenFlow qui se trouvent sur le chemin de placement de chaque portion.

6.2 Limitations de la solution proposée

Malgré le fait que nous avons obtenu de bons résultats en testant notre outil, notre travail présente cependant quelques limites :

Le calcul de routage nécessaire à la mise en oeuvre des aspects de la garantie et de la limitation de la bande passante et des politiques de compositions de points d'acheminement se base sur la résolution d'un problème d'optimisation exprimé moyennant un programme linéaire en nombres entiers. Le temps nécessaire à une telle optimisation, est assez long pour de grands réseaux ce qui ne permet pas de s'adapter rapidement aux changements des requis des utilisateurs du réseau. De ce fait, nous prévoyons définir une approche rapide qui permet de déterminer le routage adéquat en se basant sur des algorithmes d'approximation (Gonzalez, 2007) ou sur des heuristiques.

Notre outil n'offre pas du support d'un langage qui permet de spécifier les requis qui concernent la bande passante et les politiques de compositions des points d'acheminement. De ce fait, nous pourrions définir notre propre langage ou utiliser un langage existant ce qui nécessitera la définition d'une grammaire qui permettra de générer un analyseur syntaxique et sémantique pour pouvoir assimiler les requis.

Dans le cadre de placement des règles, l'algorithme glouton et l'algorithme basé sur la recherche de 'Min Cost Max Flow' utilisent un graphe qui lie les portions aux commutateurs qui se trouvent sur leurs chemins de placement. Le nombre de noeuds représentant les portions peut être très élevé quand les règles sont définies sur plusieurs dimensions. De ce fait, notre outil peut consommer beaucoup de mémoire pour représenter ce graphe.

6.3 Améliorations futures

Notre outil permet de générer les configurations nécessaires pour mettre en oeuvre les aspects de gestion considérés. Ces configurations incluent le routage et les associations entre les commutateurs et les règles. Cependant, notre outil ne permet pas d'installer ces configurations directement dans les commutateurs compatibles OpenFlow. De ce fait, nous prévoyons l'in-

tégration de notre outil dans l'un des contrôleurs existants tels que Floodlight(Ryan, 2015) et Opendaylight(ope, 2015) qui sont basés sur Java et qui permettent de communiquer avec les commutateurs à travers une API OpenFlow.

D'autre part, dans le cas où la capacité disponible dans le réseau ne permet pas de placer toutes les règles, nos approches essaient de placer le maximum des règles. Les règles non placées sont mises dans le contrôleur qui s'occupe de leurs installations en cas de besoin. Cependant, vu que les commutateurs sont saturés alors le contrôleur ne peut pas placer aucune nouvelle règle. Ceci oblige à tous les flux arrivant dans le réseau et dont les règles correspondantes ne sont installées en aucun commutateur de passer obligatoirement par le contrôleur. Dans ce cas, on prévoit définir une approche qui permet de gérer les règles non placées en substituant ces règles par les règles placées les moins utilisées. La fréquence d'utilisation de chaque règle placée peut être déterminée moyennant les compteurs offerts par les commutateurs compatibles OpenFlow.

Bien que notre outil met en oeuvre trois aspects de gestion importants pour les réseaux SDN, nous pourrions l'étendre pour supporter d'autres aspects également importants, à savoir la virtualisation et le partitionnement des réseaux. Le support de ces aspects par notre outil permettra d'avoir un environnement complet pour faciliter la gestion des réseaux SDN.

RÉFÉRENCES

"Opendaylight controller", 2015. En ligne : `https://wiki.opendaylight.org/view/Main_Page`

S. Agarwal, M. Kodialam, et T. Lakshman, "Traffic engineering in software defined networks", dans *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 2211–2219.

K. Beck, *Test-driven development : by example*. Addison-Wesley Professional, 2003.

P. Ben, L. Bob, H. Brandon, B. Casey, B. Curt, C. Dan, T. Dan, M. David, W. David, C. Edward, et G. Glen, "Openflow switch specification", June 2012. En ligne : `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf`

D.-S. Chen, R. G. Batson, et Y. Dang, *Applied integer programming : modeling and solution*. John Wiley & Sons, 2010.

I. I. CPLEX, "V12.1 : User's manual for cplex", *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

G. B. Dantzig, A. Orden, P. Wolfe *et al.*, "The generalized simplex method for minimizing a linear form under linear inequality restraints", *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 183–195, 1955.

M. Ehrgott, "A discussion of scalarization techniques for multiple objective integer programming", *Annals of Operations Research*, vol. 147, no. 1, pp. 343–360, 2006.

M. Ehrgott et X. Gandibleux, "A survey and annotated bibliography of multiobjective combinatorial optimization", *OR-Spektrum*, vol. 22, no. 4, pp. 425–460, 2000.

L. R. Ford et D. R. Fulkerson, "Maximal flow through a network", *Canadian journal of Mathematics*, vol. 8, no. 3, pp. 399–404, 1956.

N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, et D. Walker, "Frenetic : A network programming language", dans *ACM SIGPLAN Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

J.-C. Fournier, *Graphs Theory and Applications : With Exercises and Problems*. John Wiley & Sons, 2013.

M. Gabay, H. Cambazard, et Y. Benchetrit, “A reduction algorithm for packing problems”, 2014.

F. Giroire, J. Moulierac, et T. K. Phan, “Optimizing rule placement in software-defined networks for energy-aware routing”, dans *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE, 2014, pp. 2523–2529.

R. Gomory, “An algorithm for the mixed integer problem”, DTIC Document, Rapp. tech., 1960.

T. F. Gonzalez, *Handbook of approximation algorithms and metaheuristics*. CRC Press, 2007.

N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, et S. Shenker, “Nox : towards an operating system for networks”, *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

N. Kang, Z. Liu, J. Rexford, et D. Walker, “Optimizing the one big switch abstraction in software-defined networks”, dans *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 13–24.

Y. Kanizo, D. Hay, et I. Keslassy, “Palette : Distributing tables in software-defined networks”, dans *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 545–549.

P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, et S. Whyte, “Real time network policy checking using header space analysis.” dans *NSDI*, 2013, pp. 99–111.

A. Khurshid, W. Zhou, M. Caesar, et P. Godfrey, “Veriflow : verifying network-wide invariants in real time”, *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.

M. Klein, “A primal method for minimal cost flows with applications to the assignment and transportation problems”, *Management Science*, vol. 14, no. 3, pp. 205–220, 1967.

A. H. Land et A. G. Doig, “An automatic method of solving discrete programming problems”, *Econometrica : Journal of the Econometric Society*, pp. 497–520, 1960.

S. S. Lee, K.-Y. Li, K.-Y. Chan, Y.-C. Chung, et G.-H. Lai, “Design of bandwidth guaranteed openflow virtual networks using robust optimization”, dans *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE, 2014, pp. 1916–1922.

H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. Godfrey, et S. T. King, "Debugging the data plane with anteater", *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 290–301, 2011.

G. Mavrotas, "Effective implementation of the ε-constraint method in multi-objective mathematical programming problems", *Applied mathematics and computation*, vol. 213, no. 2, pp. 455–465, 2009.

N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, et J. Turner, "Openflow : enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

B. Meindl et M. Templ, "Analysis of commercial and free and open source solvers for linear optimization problems", *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, 2012.

M. Moshref, M. Yu, A. B. Sharma, et R. Govindan, "Scalable rule management for data centers." dans *NSDI*, 2013, pp. 157–170.

X.-N. Nguyen, D. Saucez, C. Barakat, et T. Turletti, "Optimizing rules placement in openflow networks : trading routing for better efficiency", dans *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 127–132.

Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, et M. Yu, "Simple-fying middlebox policy enforcement using sdn", dans *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.

K. A. Ravindra, T. L. Magnanti, et J. B. Orlin, *Network flows : Theory, algorithms, and applications*. Prentice Hall Englewood Cliffs, 1993.

I. Ryan, "Floodlight controller", 2015. En ligne : `https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+Documentation`

R. Sarwar et L. David, "Northbound interfaces", June 2013. En ligne : `https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf`

M.-K. Shin, K.-H. Nam, et H.-J. Kim, "Software-defined networking (sdn) : A reference architecture and open apis", dans *ICT Convergence (ICTC), 2012 International Conference on*. IEEE, 2012, pp. 360–361.

R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, et N. Foster, “Merlin : A language for provisioning network resources”, dans *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 213–226.

X. Wang, W. Shi, Y. Xiang, et J. Li, “Efficient network security policy enforcement with policy space analysis”, *IEEE/ACM Transactions on Networking*, 2014.

M. Yu, J. Rexford, M. J. Freedman, et J. Wang, “Scalable flow-based networking with difane”, *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 351–362, 2011.

Y. Yuan, R. Alur, et B. T. Loo, “Netegg : Programming network policies by examples”, dans *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. ACM, 2014, p. 20.

E. W. Zegura, K. L. Calvert, et S. B. Acharjee, “How to model an internetwork”, dans *INFOCOM’96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, vol. 2. IEEE, 1996, pp. 594–602.

S. Zhang, F. Ivancic, C. Lumezanu, Y. Yuan, A. Gupta, et S. Malik, “An adaptable rule placement for software-defined networks”, dans *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 2014, pp. 88–99.